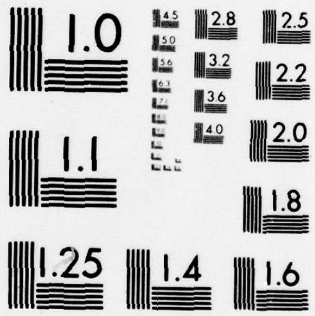
AD-A074 205 FORD AEROSPACE AND COMMUNICATIONS CORP PALO ALTO CA F/G 9/2
PRELIMINARY DESIGN OF A NETWORK VIRTUAL DATA BASE SYSTEM.(U)
AUG 79 F30602-77-C-0158
UNCLASSIFIED RADC-TR-79-102 NL

1 OF 2
AD
A074205


1.0
4.5
5.0
5.6
6.3
7.1
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

LEVEL II

12

RADC-TR-79-102
Final Technical Report
August 1979

PRELIMINARY DESIGN OF A NETWORK VIRTUAL DATA BASE SYSTEM

Ford Aerospace & Communications Corporation

Laboratory Directors' Fund No. 01737709

PRELIMINARY DESIGN OF A NETWORK VIRTUAL DATA BASE SYSTEM

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

RADC-TR-79-102 has been reviewed and is approved for publication.

APPROVED: Andrew S. Kozak

ANDREW S. KOZAK
Project Engineer

APPROVED: [illegible]

ROSS H. ROGERS, Colonel, USAF
Chief, Intelligence & Reconnaissance Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

This effort was funded totally by the Laboratory Directors' Fund.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE (19)		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) RADC-TR-79-102	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) PRELIMINARY DESIGN OF A NETWORK VIRTUAL DATA BASE SYSTEM,		5. TYPE OF REPORT & PERIOD COVERED (9) Final Technical Report,
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) N/A		8. CONTRACT OR GRANT NUMBER(s) (15) F30602-77-C-0158
9. PERFORMING ORGANIZATION NAME AND ADDRESS Ford Aerospace & Communications Corporation 3939 Fabian Way Palo Alto CA 94303		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61101F (16) 01737709 (17) 77
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (IRDE) Griffiss AFB NY 13441	(11)	12. REPORT DATE August 1979
		13. NUMBER OF PAGES 151
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same (12) 151p.		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Andrew S. Kozak (IRDE)

This effort was funded totally by the Laboratory Directors' Fund.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Data Processing
Network Processing
Data Base Transparency

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Development of the Network Virtual Data Base System (NVDBS) and the Adaptive User Interface (AUI) address several needs of DOD intelligence systems. The NVDBS could provide a bridge between different data base management systems (DMBSs) on different host computers (nodes) linked through a network, without being required to know how to access each DBMS directly. The network level description of the information and its logical structures would be interactively available to the casual user. The informed user could then "navigate"
(cont'd)

over

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

389 852

mt

Item 20 (Cont'd)

through the Network Data Model (NDM). Further, the analyst could discover and link correlated data, for the NVDBS will support "virtual linkages" between records physically located on different DBMSs of the NVDBS network. For a DBMS to be accessible on the NVDBS network, it must be on a node of the network, and must be linked to the NVDBS through an interface tailored to the idiosyncrasies of the particular DBMS and node. The NVDBS will behave as an ordinary user of each DBMS on the network. The NVDBS also will support the creation of "virtual linkages" between record instances physically located on different hosts. This report overviews the preliminary design study.

CONTENTS

EVALUATION

A Data Base Management System (DBMS) is a software package which runs on a computer in order to manage data, i.e. perform updates, retrievals, deletions and additions on data or collections of data.

A computer network is composed of nodes which are linked through communications lines and communication processors. In this report, each DBMS is assumed to be on a separate node of a computer network. The purpose of the report is to present some preliminary design parameters of an ambitious concept: the Network Virtual Data Base System (NVDBS). Such a system could provide access to a set of different DBMSs, and represents a user interface to such a set of DBMSs.

A given DBMS contains and maintains its own Data Base. Such a Data Base is the collection of all data which has been defined in the DBMS, and can be accessed and maintained through the DBMS only. The problem is to provide a capability for the user to access all the different Data Bases of the various DBMSs which are on a network, and do so in a manner which is the most convenient for the user.

A major convenience is to speak the same language to all the different DBMSs on the network rather than have to learn the different languages. This would happen if all DMBSs were identical systems (homogeneous network) which handled different Data Bases. But that is not the case; in reality, there are different computers and different systems on most networks. Another convenience (or rather an improvement on the previous convenience) is to be able to speak to all DMBSs simultaneously, and ask global questions from the network of DBMSs.

Thus, the goal of NVDBS is to provide a network level interface to all DMBSs, so that the user deals with one integrated system and does not have to keep track of which DBMS has which data. The virtue of such an approach is that the NVDBS builds upon existing DBMSs -- which required substantial investments and which are operational -- rather than starting from scratch. The main feature of the NVDBS is thus the transparency that it could provide the user. The user's view is no longer several Data Bases, but rather one single Data Base, called the Network Virtual Data Base: an integration of the different DBMSs Data Bases performed through software at the logical level. In principle, some related tasks could be handled by the NVDBS. For instance, the NVDBS could be used for maintenance of duplicates and recovery of nodal Data Base(s), if there is enough redundancy built into the Virtual Data Base to do so.

Section 0 provides the motivation and goals of the NVDBS. Section 1 is an overview of the principles of the NVDBS, and includes a description of the essential components. Section 2 presents the Man-Machine interface mechanism, concentrating on the usage and user oriented features. The mechanism makes four basic modes available to the user. Under one mode, the user may navigate through a directory and when the desired headings or data types are found, the user may switch to a query mode, e.g. to retrieve certain information (data) about the types in question. Another mode allows the local manipulation of the network-retrieved data and yet another mode allows the virtual "linkage" of different data types. Section 3 covers the mechanism for virtual linkages between data types located on different nodes. Section 4 exposes the relational -- i.e. tabular -- information system which is used for local data manipulation.

Section 5 presents a concurrency control mechanism (to maintain a consistent view of the data) and a deadlock avoidance scheme, together with some crash recovery procedures.

[illegible]

ANDREW S. KOZAK
Project Engineer

PRELIMINARY DESIGN OF A NETWORK VIRTUAL DATA BASE SYSTEM

Software Technology

Ford Aerospace

0. INTRODUCTION AND SUMMARY

0.1 DISTRIBUTED DATA BASES

The Network Virtual Data Base System (NVDBS) and the Adaptive User Interface (AUI) address several critical needs of DoD intelligence systems. The first of these needs is a method of accessing disparate data bases located on dissimilar computers. These data bases are usually dedicated to one particular form of collected intelligence data, and are geographically distributed. The US Military Intelligence network in Europe is a good example. ELINT data is stored and maintained by USEUCOM in Stuttgart; ground order-of-battle information for the army is maintained by USAREUR in Heidelbert and must be accessed through the USEUCOM communications computer, the Combat Operations Intelligence Center (COIC) at Ramstein maintains the air and missile order-of-battle and requires electronic order-of-battle information from USEUCOM; the 497th Reconnaissance Technical Group (RTG) provides PHOTINT information for

*This work was supported in part by the Rome Air Development Center, Griffiss AFB, NY, under contract number F30602-77-C-0158, and in part by Ford Aerospace and Communications Corporation Independent Research and Development Projects.

the theater along with target/defense analysis information; the Tactical Fusion Center provides near-real-time updates to the air and missile order-of battle files and so requires tight coupling with the data files in the COIC; the TFC also requires all source information for the Commander in Chief of European forces during crisis and wartime situations. This network is illustrated in Figure 1.

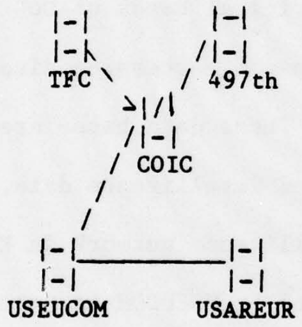


Figure 1. USAFE Intelligence Network

This distributed data base capability is not currently available in the European theater. Rudimentary message traffic between the elements of the system is provided by the Standard Software Base (SSB). The SSB provides analyst with the ability to query the databases on the network, assuming the format of the commands of the relevant Data Base Management System and the different access permissions are known. The design of a common query language facility is progressing towards interactive analyst support [QIP77].

0.2 MOTIVATION FOR THE NVDBS

The motivation for developing the NVDBS concept is the need for text image integration at the level of the intelligence analyst. This is conceived as a need to broadly associate or link the different pieces of data which are collected on different DBMSs. These pieces are usually called record occurrences [CODASYL-DBTG71]. At this point in time, the need for the capability of low level record access to the data bases in the theater is real. Why is this low level access required if a common network query approach is available? The primary advantage of restricting ourselves to this primitive form of access is that it represents a well defined and common first step for the first phase of an NVDBS implementation. All other refinements can be built on or benefit from the preliminary implementation and experience of the record level access. It should also be noted that one of our guiding principles is to keep the NVDBS-to-DBMS interface simple. This is possible at this stage because all operational DBMSs provide for record accesses.

There are also several long range considerations for chosing this approach. Work is being done in the area of powerful relational queries and their relational decomposition into single relation queries [WONG] [YAO], and the use of different geographic clusters [INGRES78]. This relational effort builds easily on the record level approach of the NVDBS.

A second long term consideration is the requirement to support situation displays and monitoring interfaces for all users of the intelligence network, especially the tactical units. The NVDBS record level approach can ultimately provide the unique capability of monitoring changes in the distributed databases as they occur. This is possible because most of the information is organized on a record level, by target or geographic area. This change information could be used to update AUI supported textual, graphical and image data at the tactical unit. The records or areas monitored would depend upon the tactical unit combat area of interest. This area could be dynamically modified by the intelligence officer using the AUI. Thus the intelligence officer specifies a list of targets or an area of interest that will be monitored for changes in each of the databases specified. Current situation data could be kept at the local AUI so that only change data need be transmitted on the network.

0.3 DATA CORRELATION

Inherent in the ability to access a distributed data base system is the ability to automate some of the basic processes involved in the implementation of the intelligence data correlation process. Correlating is itself a rather vague process of taking dissimilar sets and sources of information and producing an integrated picture of a situation or event. The problem in trying to aid in the performance of the correlating process is knowing both the availability and the validity of certain information and determining what information is relevant or pertinent to the situation being examined. Representation of the data in the intelligence network and the relationships between data records is the function performed by the Network Data Model (NDM). The NDM provides the analyst with an integrated graphic representation of the typical data in the network and the logical paths that connect the data types. The analyst can use the NDM to navigate through the network while retrieving and examining individual record instances as he travels along. An example of this navigational aid in the correlating process is illustrated by an analyst's attempt to find the source of a new signal.

> The only information the analyst has is an ELINT report on a new frequency intercept, the days it occurred on, and the fact that the signal is at the same geographic location as a known target. The analyst finds by examining the Automated Installation File (AIF) that the identified target location is a radar site. Associated with this target information in the NDM are ELINT records containing transmission and reception frequencies; HUMINT records with reports of the radar facility, its construction, and how many people work there; SIGINT records that describe communications between the radar station and the central command unit; and finally PHOTOINT data of the radar installation and the surrounding area.
>
> The analyst might first choose to determine what ELINT information is available for the intercepted frequency and for the radar installation. The NDM may show him that there are records in the ELINT data files that are organized on a frequency range basis and give descriptions and characteristics of known devices that exist in these frequency ranges. The NDM also shows him that EOB data is available for the radar installation and the frequency ranges and characteristics that are associated with that site. By retrieving certain records of ELINT data he is able to determine that the intercept frequencies have not previously been associated with the radar site and that the only known devices emitting in that frequency range are a new enemy laser weapon and the electronic amplifiers of an internationally known punk rock band. Being a secret punk rock fan he knows this group has never been in the target area and so determines to concentrate on the laser weapon. Using the NDM the analyst finds that HUMINT files have subject-related search files. A retrieval of records associated with the laser weapon contains a report of a laser scientist being observed in the target area.
>
> The analyst decides to next examine the PHOTINT records associated with the radar installation. The NDM shows that PI exploitation records are organized on BE number and by geographic location. Associated with each target record is also a history of coverage and all previous reports of imagery of the installation. The history of coverage record shows that a photo of the radar installation was taken on one of the days that the new frequency was intercepted. He requests this photo and the next earlier photo taken three weeks prior to the intercept photo. From examination of the photos he observes an unusual difference in concentration of cars around a warehouse a block from the radar site. A magnified photograph of the parking lot reveals the laser scientist's car. And so a new BE number is born.

This fictional example illustrates "intelligence fusion", i.e. the use of the NDM and the NVDBS by the analyst to navigate through the different data files in the intelligence network to build an integrated understanding of an event. The significant advantage of the NDM and navigating approach is that the analyst has a constant help available to express the required names in the different commands he transmits to the system. Furthermore, the system - under the NDM scanning mode - will remember what are the current types in which the analyst is interested, and will fill them in as default when these are not respecified. Lastly, the analyst is aware of the latest changes in the integrated structure, and he is able to ignore the different origins of the data presented in the NDM and constituting the Network Virtual Data Base. In fact he may not know or may not be authorized to know which are the DBMSs (components of the NVDBS) supporting such data.

0.3.1 Image text integration

The methodology for image text integration involves two major capabilities. One is to access the various data bases that contain image and textual elements; the second is to provide the mechanism for combining these different types of data for display and manipulation by the user. The Network Virtual Data Base System provides the distributed data base access capability using record level processing and the Network Data Model to define the relations between data in the network. The Adaptive User Interface provides the integration of image and textual information using tabular algebra for display and manipulation of the various data types.

0.4 ORGANIZATION OF THE NEXT 6 SECTIONS

Section 1 gives an overview of the principles and main components of the NVDBS.

Section 2 presents the main features of the Adaptive User Interface, with its four different modes. Emphasis is on the mode to scan the directory of the NVDBS, and the touch panel/soft keys approach to the man/machine dialogue.

Section 3 covers the manner in which the virtual linkages between data located on different nodes are in fact implemented; emphasis is on the manner in which integrity and consistency may be maintained.

Section 4 exposes the relational system for local data manipulation; the elementary structure is the table. The whole section is relevant to the data manipulation mode of the AUI, and in a later phase, to some common relational query language.

Section 5 discusses the concurrency control mechanism to prevent conflicts between NVDBS transactions.

0.5 THE DIFFERENT PHASES OF THE NVDBS PROJECT

Phase 1 is the preliminary design of the NVDBS: particularly the basic principles, approaches, important algorithms and procedures. This report marks the end of phase 1, before its validation. The authors seek comments, criticisms and suggestions.

Phase 2 will be the design and implementation of a prototype system oriented towards the hierarchical or network class of DBMSs, with only a basic record access query mode. The purpose is to achieve some network integration of the multiple DBMSs hosted on a computer network, at a modest cost.

Phase 3 will extend the relational system -- initially designed as a local data manipulation mode for the user and as an access to the Network Data Model -- in order to handle queries relative to several record occurrences and/or set occurrences. Essentially, this third phase would coincide with the advent of more relational DBMSs in operation in the field. The result sought by the third phase is the maximal download of the query treatment to the different DBMSs of the network. This download should be accompanied by faster response time because of the minimization of network communications.

This phase 3 will require the implementation of relational front-ends on top of the non relational DBMSs of the network.

A phase 4 will be concerned with optimization [BLASGEN76].

1. GENERALITIES

1.1 NVDBS PRINCIPLES

This section defines the basic operating principles of the Network Virtual Data Base System and the modules that compose it. The Network Virtual Data Base System (NVDBS) provides a bridge between different data base management systems (DBMS) on different host computers, known as nodes, linked through a network. A user accesses the different databases on the network through the NVDBS, and does not know how to access these DBMSs directly. Only the network level data description of the logical information structures available need be known to the casual user. To be accessible by the NVDBS, a DBMS must be on a node on the network, and must be linked to the NVDBS through a DMSREP, the interface tailored for the particular DBMS and node.

The first principle of the NVDBS is not to require major changes in any DBMS. More precisely, the NVDBS acts as an ordinary user and does not require special treatment. This principle minimizes the impact of the network on the DBMSs and their nodes. In particular, the interface between the NVDBS and a specific DBMS, the DMSREP, should be as small and simple as possible.

The second principle is to present the user with a simple, integrated image of the data on the network. The casual user does not know which DBMS has the data he wants. To this end, the implementation of "virtual linkages" involves the maintenance of information at the network level. This information allows the NVDBS to easily associate data from a DBMS on one node with other data in a DBMS elsewhere, without placing an undue burden on host computer or DBMS. In principle, this means that the "virtual linkages" should be defined logically, without any assumption relative to any specific record occurrences. Namely, algorithms are described through "RECIPES" which are canned routines to actually accomplish the linkage in both directions.

A last observation. The NVDBS is not a panacea. Rather, the NVDBS ambitiously aims at a very specific objective: providing an integrated view of different data bases on heterogeneous systems, and providing a record level access to the network. Admittedly, such a record-at-a-time system will not be efficient for heavy use. But heavy use will almost always be local, on an individual DBMS. Other efforts are concerned with the decomposition of multi-variable queries and with the optimization of their treatment in a distributed data base environment [WONG76,77]. These efforts concentrate on the efficient treatment of complex queries, but require either a single distributed DBMS or a homogeneous network formed of identical DBMSs. The NVDBS does not.

1.2 OVERVIEW

The NVDBS is a system which should appear as an ordinary user to each DBMS on a host of the network. The DBMS itself is not modified in order to support the NVDBS. To achieve this purpose, the DMSREP is a module installed on the host computer to

a. serve the particular transaction received from the NVDBS and intended for the particular DBMS to which the NVDB-DMSREP is assigned,

b. transmit the answer of the DBMS to the rest of the NVDBS, after adequate treatment.

The NVDBS interface sequence is depicted in Figure 2. A user accesses the NVDBS through an Adaptive User Interface (AUI) which allows simple interactions geared towards a casual user rather than a programming expert.

```
 man/machine dialog                                         |--------|
 ---------                                            |--->|DMSREP#1|
 |        |                                           |    |--------|
 |        V                                           |
 |     |-----|               |-----|                  |    |--------|
 ---->| AUI |<------------>|AGENT|<-------------|--->|DMSREP#2|
       |-----|               |-----|                  |    |--------|
          ^                     ^                     |
          |                     |                     |    |--------|
          |                     |                     |--->|DMSREP#3|
          |                     |                          |--------|
          |       |-----|       |
          |---->| NDM |<------|
                  |-----|
```

Figure 2. NVDBS Overview

The AUI eventually formulates the user's query and passes it to the AGENT. In turn, the AGENT processes the query, parcels out the various subqueries to the different DMSREPs, and returns the results to the AUI. Both the AUI and the AGENT call on the Network Data Model (NDM) to obtain information about the Network Virtual Data Base. The AUI wants to know the logical view of the unified Data Base. The AGENT wants more details about "which data in which DBMS on which host is really implicated in a particular query transaction". The AGENT accomplishes the analysis and decomposition of the user's query. Eventually, the AGENT formulates a sequence of NVDBS commands composed of verbs and arguments. This series of commands is broken down into transactions directed to different DMSREPs located on different nodes. These DMSREPs are the network interfaces to the DBMSs which manage the data relevant to the users' queries. The DMSREP receives and processes a transaction by translating the NVDBS commands into the particular DBMS language. It also processes the answers and returns the results of the whole transaction to the AGENT which had mailed it in the first place. At this point, the AGENT puts together the different answers obtained from the different DMSREPs involved. This synthesis leads to an answer which is then sent back to the AUI. The AUI formats it on the CRT for the user, or passes it to the user's process.

1.3 THE AGENT

The main functions of the AGENT are:

- to interface with the NDM in order to make all different levels of DBMS and network linkage information transparent to the AUI,
- to execute algorithms for the analysis of a transaction and the required "exploratory" phase when the transaction encompasses updates and possible consistency problems,
- to execute algorithms for decomposition of an AUI transaction into different AGENT subtransactions directed to different DMSREPs,
- to execute algorithms to collect the results from the different DMSREPs and synthesize the AUI transaction's result.

Some functions of the AGENT are described in more detail in later sections.

1.4 THE DMSREP

The design of a specific DMSREP will heavily depend upon the DBMS with which it interfaces, the available front ends for the DBMS, and the underlying operating system. Clearly, a DMSREP transaction can open and close a particular subschema and represent a DBMS session all by itself.

The DMSREP must create a "multi-thread" environment,i.e. it needs to concurrently process transactions from multiple users. In some systems it may be possible to interface a multi-thread front end to the DBMS. In other cases, it may be necessary to create the multi-thread environment within the DMSREP.

When retrieving a record occurrence, the DMSREP will return a description of the record. This description will include field types, names, lengths, and modification protection flags. The modification protection flags will be used to prohibit the AUI from modifying sacrosanct fields in the record, such as embedded keys.

The DMSREP will also keep track of authorization to modify the associated DBMS. The authorization will specify whether a user can read, modify, and/or delete a record.

One interesting aspect in the design of the DMSREP is the trade-off between the optimization of the DMSREP-role with respect to the DBMS and the host on one hand, and the response time of the DMSREP with respect to AGENT on the other.

1.5 THE NETWORK DATA MODEL

1.5.1 NDM Principles: The majority of DBMSs used today are of a record-oriented type, either hierarchical in nature , such as [IMS], or plex-structured like [CODASYL-DBTG71]. Therefore, the basic info-structure for the NVDBS is patterned after the "owner-member set" [DATE75]. This orientation was chosen for reasons of practicality. Although relational model based systems [CODD/DATE74] have much to recommend them from a theoretical standpoint, there is presently no large scale relational model-based systems in operational use. Further, it can be shown that a relational model-based system can be described as an owner member set model system. The NVDBS data structure is based on two concepts: records and sets. The NDM characterizes the ensemble of participating DBMSs in terms of these concepts. In essence, the NDM is a data base schema for the entire network. The schemata of the individual participating DBMSs can, thus, be thought of as being sub-schemata of the NDM. As explained in [CHU76] [PEEBLES78], the NDM would be distributed (mostly replicated) in the NVDBS in view of a very low update expectation. For optimal performance, NDMs should be close to both AUI and AGENT.

1.5.2 Graph model It is useful to present a directed graph model for the information structure because the user benefits from an immediate picture [BACHMAN69]. The definition of the graph model follows.

- There are two kinds of nodes: a record node and a set node.
- Each node has a name : a record name or a set name.
- Each set node has one or more outgoing edges.
- Each set node has one or zero incoming edge.
- A set node points to one or more record nodes.
- A record node points to zero or more set nodes.
- A record (/set) node cannot point to a another record (/set) node.

Note the meaning of a record node as a record type, and the fact that not all occurrences of a record node may necessarily belong to a set, but all occurrences of an owner record node belong to the owned set.

1.5.2.1 AUI Representation of the NDM

The AUI will give the user a general view of the NDM, i.e. essentially the graph model described above. Figure 2 shows an example of the digraph with "R" (record) and "S" (set) nodes. A graph edge leading into a set node from a record node represents the owner of the set. The edges leading from the set node to other record nodes represent the member records. Note that this model does not account for the fact that some record occurrences may not belong to a set of which the record is member.

```
|-------------------------|
|(n1)   (n2)   (n3)   (n4)|
| a      b      d      d  |
| |      |      |      |  |
| ---   ---     |      |  |
| | |   | |     |      |  |
| b c   a d     c      a  |
|-------------------------|
```

(I) owner-member sets

```
|-------------------------------|
|     <-------S(n4)<-------     |
| R(a)<-------S(n2)------->R(d) |
|  |            ^           |   |
|  |            |           |   |
|  V            |           |   |
| S(n1)------->R(b)         V   |
|      ------->R(c)<--------S(n3)|
|-------------------------------|
```

(II) graphic representation

Figure 3. Network Data Model - Representation of info-structure

The user may then point to each node to obtain a description of the set or record type. The names of the records and sets are network names. This is required to eliminate duplicate names. The network naming convention both shields the user from different DBMS naming conventions and enforces naming standardization at the network level.

1.5.3 Overview of the NDM

1.5.3.1 The Recipes:

In principle, the NVDBS should define record occurrence(s) in a logical manner, a manner which cannot become obsolete as deletions or insertions of record occurrences are made in the different DBMSs. This logical definition of the record occurrences is described in algorithms coded into "Recipes". These recipes are nothing but a sequence of commands to actually get to the record occurrences without referring to physical record occurrences or data base keys. These recipes guarantee the transparence of such operations as ERASE, STORE, CONNECT, and DISCONNECT, as they are carried by the individual DBMSs in an independent manner.

To be fair, however, there are several circumstances in which this high level of independence would be unjustifiably expensive. For instance, there may be a knowledge of low frequency of update at fixed intervals, or there may be a mechanism to notify the users of these updates (date of last update), or there might be a situation where all updates are made through the NVDBS, thereby ruling out any need for expensive recipes. In all these instances, it is reasonable to use keys to define record occurrences. The assumption that all DBMSs on the network be similar to the [CODASYL-DBTG71] model implies that when a record occurrence is first created, and stored in a particular DBMS, this occurrence has been allocated a Data-Base-Key (DBK). A non CODASYL system such as [ADABAS74] has internal sequence numbers with the desired properties of the DBK. Such a DBK does not change for the whole life of the record occurrence, in spite of data base reorganizations and updates. This is an important condition which must somehow be respected. Another is that the DBK value be not reused, after a deletion. In the case when a DBMS on the network does not have the properties of the DBK, there must be an appropriate interface and index files incorporated in the DMSREP mechanism, if performance demands and circumstances allow the use of DBKs instead of Recipes.

1.5.3.2 The VS:

The Recipes allow the NDM to maintain "virtual sets" (VSs) associating different record types existing on different DBMSs. The most important restriction to the generality of "virtual linkage" [McCauley77] is the following:

a particular record type belongs to one and only one DBMS.

This means that all record-types involved in a VS are precisely record types existing on different DBMSs on the network; all occurrences of a record type are therefore located on only that DBMS where that record type is defined. There is no such thing as a record type having several "copies" of its occurrences in different DBMSs. The NVDBS or its user could always recognize the similarity between two different record types located on different DBMSs and could create a VS to associate these records. There are several reasons that virtual record types were not also defined in the NVDBS. Primary among the reasons was the implied complexity and power that would be required in the NVDBS to maintain and enforce the formats and use of these virtual record types among the many disparate users of the network. This would violate the first principle of the NVDBS that it is merely a user of the DBMS on the network. An owner record type of a VS must exactly map into a specific DBMS record type. The same is true for each VS member.

The sets of a particular DBMS are referred to as real sets (RS). RSs are represented "as is" by the NDM, except for specific names which may be modified because of naming conflicts. The NDM maps unique "NVDB-names" onto the names of sets, record types, in the different DBMSs. One possible choice (but not a very attractive one) for "NVBD-names" which would guarantee uniqueness is the concatenation:

/host-name/DBMS-name/Set-name or record-name

for a set or record. Of course, a network VS-name does not map into anything since it exists only at the network level. The VS and its membership options are the only information which is carried strictly at the network level. The Table of Record Names (TRN) and the Table of Set Names (TSN) contain such information as the last update date of the DBMS of which the NDM is aware. This helps the NDM decide to check the schema of a DBMS for updates, and sometimes also the DBKs involved in the VSs.

One table, the Real Set Table (RST) contains the description of sets which are entirely contained in a given DBMS, together with the set membership options (SMO).

```
|-------------------------------------|
|  TRN                                |
|  |------|------|-----|-----------|  |
|  |NVDBS |DBMS  |DBMS/|Other      |  |
|  |record|record|HOST |Information|  |
|  |name  |name  |     |           |  |
|  |------|------|-----|-----------|  |
|  | .etc.|...   |...  |...        |  |
|                                     |
|  TSN                                |
|  |------|------|-----|-----------|  |
|  |NVDBS |DBMS  |DBMS/|Other      |  |
|  |set   |set   |HOST |Information|  |
|  |name  |name  |     |           |  |
|  |------|------|-----|-----------|  |
|  | .etc.|...   |...  |...        |  |
|                                     |
|-------------------------------------|
```

Figure 4. TRN and TSN Table formats

Another table, the Virtual Sets Table (VST) contains the description of virtual sets. One entry describes a VS and its SMO, and points to the Recipes for getting owner or member occurrences. These Recipes driven properly create the illusion of a table, called the Virtual Set Occurrences Table (VSOT). The VSOT may be a real table when DBKs are used in place of Recipes, i.e. when there is no possibility of updates of record occurrences involved in the VS. Otherwise, the VSOT happens to be a virtual table; and the recipes recreate dynamically the data contained in the VSOT whenever needed. The VSOT associates the record occurrences characterized by their DBK's.

```
|-----------------------------------|
|RST, typical entry, varying length |
|                                   |
|   NDM-set-name                    |
|   NDM-owner-name                  |
|   NDM-member-name#1               |
|   SMO#1                           |
|   NDM-member-name#2               |
|   SMO#2                           |
|   ...etc.                         |
|                                   |
|VST, typical entry, varying length |
|                                   |
|   like RST typical entry, but with|
|   additional field to point to    |
|   associated VSOT                 |
|                                   |
|-----------------------------------|
```
Figure 5. Virtual Sets Table (VST) Contents

Each VST entry contains a pointer to the corresponding VSOT table. For a given VS, the VSOT associates the DBK of an occurrence of the owner record with the DBK values of the occurrences of each member record for a particular set occurrence. In general, though, such a VSOT table does not exist as such, but is virtually available through the execution of code (Recipe) stored in the VSOT in lieu of the table data itself. Whether the VSOT is available statically or dynamically, the description below remains valid.

```
|-----------------------------------|
| VSOT-name = Set-name              |
|                                   |
| typical entry:                    |
| owner occurrence DBK value        |
| member#1 array                    |
|  member#1 occurrence DBK value    |
|  member#1 occurrence DBK value    |
|  ...etc.                          |
| member#2 array                    |
|  member#2 occurrence DBK value    |
|  ...etc.                          |
|                                   |
|-----------------------------------|
```
Figure 6. Virtual Set Occurrences Table (VSOT) Contents

The RST and the VST are often consulted by the AUI in order to satisfy the users' requests. However, the different VSOT tables, and the TRN and TSN are consulted by the AGENT in connection with the mapping of info-structures into the individual DBMSs which are members of the network.

1.5.4 <u>The Processing of Schema Changes Within the NVDBS</u>

A major obstacle that must be addressed in the implementation and operation of the NVDBS is the effect of changes in the schema of an individual database in the network schema. Because of the first principle, the NVDBS will have to react to changes after they have already occurred. The NVDBS will check a date that it maintains for the last update of each data base schema against the date of the last schema update maintained by each DBMS. If the two dates differ the NVDBS will suspend operations with that DBMS and notify the user that delays due to certain verification procedures for updates of the NDM are to be expected. The NVDBS-AGENT will automatically transmit a request for information to the DBMS in an attempt to update the network schema. If the particular DMSREP does not support that type of request or if the updated schema affects the network virtual sets, the NVDB Administrator will be alerted. The NVDBA is ultimately responsible for making and/or verifying the changes that are made to the network schema as represented in the NDM.

1.6 <u>NVDBS PROTOCOL</u>

1.6.1 <u>USERs and SERVERs</u>

Each NVDBS module may be located on different hosts, while they correspond with each other. The underlying computer network is assumed to possess a host-to-host protocol, such as the ARPANET's NCP protocol. This host-to-host protocol provides complete mailing services for a "letter" sent by one host to another host. The NVDBS protocol parses the data stream of a letter and decomposes it into transactions and their component commands (verb, arguments). The manner in which the letter is passed from the network communications module to the recipient NVDBS module is somewhat host dependent. For example under UNIX, a read issued from the recipient process could mean that the scheduler will reactivate the recipient process when the letter arrives. In general, inter-process communications and virtual buffers will provide the link between the protocol module and the NVDBS module. The inter-process mechanism is typically a function of the host operating system.

Each NVDBS module requires a USER and a SERVER, to respectively send and receive a letter. A USER translates the NVDBS commands expressed with the alphabet of the USER's host into a standard letter medium format. A SERVER module will receive the letter and decode the standard format into NVDBS commands expressed in the SERVER's host alphabet. The letters themselves are handled by the host-to-host protocol as bit strings in a transparent manner. Similarly, the USER and SERVER insulate the NVDBS from the differences between the alphabet of the hosts.

1.6.2 <u>The NP1TS and 2NTS Transmission Schemes</u>: Several approaches to the transmission of NVDBS verb sequences from the AGENT to the required DMSREPs and the response sequences from the DMSREPs to the AGENT were considered. The first transmission scheme is called the "N plus 1 Transmission Scheme"(NP1TS) [McCauley77]. The second scheme is called the "2 N Transmission Scheme"(2NTS). The two schemes are represented in Figure 2 and 3.

In the NP1TS scheme, when no recursive or stack procedures are used (no backtracking), the AGENT may address only one DMSREP (#1) which in turn will pass the rest of the transaction to another DMSREP...etc., until the whole transaction is processed. The last DMSREP to finish the processing will return the result of the transaction to the AGENT. This scheme is acceptable provided little data is involved, and the concerned DMSREPs accept the burden. The AGENT generates a simple return-socket which is enclosed with the transaction-letter mailed to the first DMSREP (#1). This socket is passed along with the transaction and results to each DMSREP involved in the query process. The last DMSREP sends the results of the transaction back to the AGENT via the return socket. The AGENT which was listening on this socket, can now start its synthesis phase. For n DMSREP's involved there will be n+1 transmissions, instead of the 2n transmissions required by having the AGENT address directly the different DMSREP's and listen for all their answers.

```
|-------------------|
| AGENT     <----|  |
|   |            |  |
|   |            |  |
|   V            |  |
| DMSREP#1       |  |
|   |            |  |
|   |            |  |
|   V            |  |
| DMSREP#2       |  |
|   ...          |  |
|   |            |  |
|   V            |  |
| DMSREP#n------>|  |
|                   |
|-------------------|
```

Figure 7. NP1TS

```
|-------------------------------|
| |<--------| NVDBS |<---------| |
| | |<------| AGENT |<-------| | |
| | |       |       |        | | |
| | |  |<---|       |<----|  | | |
|(NVDBS queries)  (DBMS answers)|
| | |  |                   |  | | |
| | |  |---> DMSREP#n --->|  | | |
| | |                       | | |
| | |------> DMSREP#2 ------>| | |
| |--------> DMSREP#1 -------->| |
|-------------------------------|
```

Figure 8. 2NTS

If much data is transmitted, the 2NTS is preferrable, because network vulnerability is greatly exposed by the NP1TS. A reference to a down DMSREP in a sequence of DBMS accesses may effectively lock a process in the NVDBS. A downed DMSREP must be brought to the attention of the AGENT so the AGENT can either report the problem to the AUI, or synthesize a different query for the AUI. In the 2NTS, the AGENT transmits to each DMSREP and is aware of the status of each DBMS involved in a query. This gives the AGENT using the 2NTS more flexibility and control and usually faster transaction times.

Let us now consider the case of backtracking. Consider a programmer who navigates in the Network Data Model. The programmer may move a cursor to an owner occurrence, to an owned member occurrence, or to a next or prior occurrence all within the owner-member set's info-structure. A record-cursor points to a particular record-type occurrence, by specifying the Host-ID/DBMS-ID/Record-name/Data-Base-Key. A Data-Base-Key (DBK) points logically to a specific record occurrence. A mapping takes place between the record cursor seen by the user (expressed in terms of network names) and the record cursor which is actually needed by the NVDBS (expressed in terms of the particular DBMS involved). This is accqmplished in part by the AGENT with NDM information and by the DMSREP which LOCATEs the DBK-value of the record occurrence. The problem arises of mapping the operations of popping up and pushing down pointers (to vertices of the graph model of the info-structure) into operations in different DBMSs [McCauley77]. The NVDBS verbs POP-CURSOR and PUSH-CURSOR allow for restoring or saving cursors for backtracking. Backtracking is especially important for data bases which may not have back pointers pointing to the prior or owner record occurrence within a set-occurrence. In the NP1TS [McCauley77], a DMSREP may be put in charge of a stack. At the bottom of the stack is a pointer back to the previous DMSREP process and its stack. When the DMSREP pops up the bottom of its stack, its stack structure is flushed, and control goes to the previous DMSREP and its stack. If however, the transaction sent by the user finishes in DMSREP#n, while leaving nonempty stacks in the different previous DMSREP's, the last DMSREP #n must flush its stack and mail back to DMSREP#{n-1} an order to flush. This order must propagate back to DMSREP#1, so as to flush all these stacks and terminate all processes of DMSREPS involved in the transaction. Thus with backtracking added, the NP1TS really demands 2n transmissions rather than n+1.

Another option is to have the AGENT keep only one stack. The transaction is then much shorter and the AGENT gets from the DMSREP the particular DBK which it will push down on the stack. The 2NTS provides additional advantages for repeat, post mortem, log, and debug operations.

```
|----------------------------|
|AGENT  <-----(answer)------ |
| |                        ^ |
| |                        | |
| V                        | |
|DMSREP#1 <--|             | |
| |          ^             | |
| |          |(flush)      | |
| V          |             | |
|DMSREP#2 -->|             | |
| |          ^             | |
| |          |(flush)      | |
| V          |             | |
|DMSREP#3 -->|             | |
| |                        | |
| |                        | |
| V                        | |
|  ----------------------->| |
|----------------------------|
```

Figure 9. NPlTS witgh backtracking

1.7 VERBS

The verbs illustrate the record level query mode for the AUI, as they are addressed to the AGENT. For the AGENT, it illustrates the Record Level Query to different DMSREPs.

In general, one should recognize that update actions requested by the AUI may give birth to many transactions in the NVDBS. For this reason, and because the NVDBS is only a user with respect to a DBMS of the network (ref. Principle #1), it is essential that the NVDBS find out whether or not ALL the intended actions in its transactions will be authorized or acceptable for any implicated DBMS on the network. Therefore, AUI transactions modifying data base on the network will give birth to two phases in the processing of that transaction by the AGENT:

a. The Exploratory Phase - analysis generation of intended commands verification of acceptability by relevant DBMSs (with exploratory transactions)

b. The Execution Phase - executing actual interaction with relevant DBMSs

The following subset of NVDBS basic verbs is reviewed in section 3:

1. BEGIN_TRANS/END_TRANS
2. INVOKE/ENDINV
3. MOVE_CURSOR NEXT/PRIOR/HEAD/TAIL/OWNER (CODASYL FIND)
4. LOCATE (with qualifiers on key)
5. PUSH_CURSOR
6. POP_CURSOR
7. READ (CODASYL GET)
8. MODIFY
9. CREATEVS name of VS (used by NVDB administrator)
10. DESTROYVS name of VS (used by NVDB administrator)
11. STATUS_INQUIRY returns status
12. DESCRIBE shows layout of data base description
13. DELETE (alone/ONLY/SELECTIVE/ALL)
14. STORE
15. DISCONNECT
16. CONNECT

The last four verbs involve the Set Membership Options. They are covered in section 2.

1.8 CONCLUSIONS OF SECTION 1

The NVDBS provides a viable low-risk alternative for the near-term requirements of distributed data management. The NVDBS is not the ultimate answer for distributed data management, as may be SDD-1 and other total system approaches. But the NVDBS does offer an approach that builds upon the large amount of manpower and money invested in current, operative DBMS. This preliminary design of the NVDBS provides basic record level retrieval capabilities. The design is structured so that more complex query capabilities and more integrity and consistency may be added later.

2. ADAPTIVE USER INTERFACE AND RECORD LEVEL QUERY

This section begins with a general description of the Adaptive User Interface's (AUI) role in the NVDBS. The basic principles used in the design of the AUI are defined and an overview of the AUI is presented. The overview includes the definition of the functional components and the general processing capacities and configuration of the AUI. The four modes of interfacing with the AUI are described. The commands of the AUI and the NVDBS_AGENT for performing record level transactions on the network are specified. Finally, an example of the user's interaction with the AUI and the commands issued to the network is given to illustrate the adaptiveness of the interface.

2.1 AUI'S ROLE IN THE NVDBS

The user accesses the NVDBS through an AUI. The AUI is an intelligent terminal connected to any host in the NVDBS. Its purpose is to allow for simple data base interactions and is geared towards the casual user rather than the programming expert. The AUI uses the logical integrated view of the DBMSs on the network, as provided by the Network Data Model (NDM), to help the user navigate through the data in the network. This AUI mode is the "NDM scanning mode".

The "Record Level Query mode" (RLQM) provides the user with the capability of retrieving or storing a record occurrence on the network. The queries are passed to a module called the NVDBS_AGENT. The AGENT determines which individual DBMSs are involved in the user's query, decomposes the user's query into subqueries (as defined by data location information in the NDM), performs query optimizations, and issues separate subqueries to the separate DBMS required to satisfy the total query. The AGENT transmits the queries as a sequence of NVDBS commands composed of network verbs and arguments in transactional blocks to a particular DBMS interface, called a DMSREP. The DMSREP is the interface between the network and a particular DBMS. The DMSREP receives and processes a transaction by translating NVDBS commands into the language of the particular DBMS it represents. The DBMS will then retrieve the specified record(s) and return the results to the DMSREP. The results of an entire transaction are assembled by the DMSREP and transmitted to the the requesting AGENT. The AGENT puts together the answers obtained from the different DMSREPs involved. This synthesis leads to an answer which is then sent back to the AUI. The AUI formats the results for display on the user's terminal, or passes it to the user's process.

2.2 PRINCIPLES

The AUI gives the user an integrated view of many disparate data bases using a network schema contained in the NDM. This network schema defines the various record types and set relationships of the network, with their options.

The AUI will present a friendly interface to the user by providing help to the user when he requests it or when the user's actions are unclear to the AUI. The AUI will also use the "interrogative" technique as defined in RENDEZVOUS [CODD74/75] to make more explicit the user's queries to the TAbular System (TAS). TAS is a relational system associated with the AUI and the NDM. TAS has powerful data manipulation commands used by both AGENT (e.g. in connection with the processing of schema changes) and AUI (e.g. data manipulation mode) in order to access the NDM. The TAS language (TAL) is based on relational algebra and is therefore close to relational calculus languages such as

[SEQUEL76] or QUEL [INGRES76,77]. Eventually, the TAS could be used as a full blown local DBMS under the data manipulation mode.

Another mode of the AUI is the administrative mode, which allows an exclusive set of users to execute certain data manipulation under TAS. These administrative commands are concerned with the creation and modification of virtual sets and other NDM changes to reflect the evolution of the data bases on the network.

In an interactive mode the AUI will use and extend the terminal transparency concept of the Terminal Transparent Display Language (TTDL) [INCO76]. TTDL provides a language for describing displays, and formats the displays for output on many different terminals. The AUI extends this concept by making the displays adaptive. For instance, as the NDM changes the associated displays will be automatically modified to present the NDM in its most current state.

The AUI will "remember" sequences of commands generated by the user in the interactive mode and run them upon request in the embedded batch mode. This implementation concept is as exposed in [LORIE77/78]. On one hand, the precompilation of the user's data statements would be a fait accompli, and the machine language code available from the NDM. On the other hand, an Access Specification Language would improve the efficiency of the system.

2.3 AUI Configuration

The physical configuration of the AUI is based upon the projected capabilities and capacities available in intelligent terminals in the 1980's. The AUI configuration is shown below.

```
                  |---------|
                  | Network |
                  |Interface|
                  |---------|
|---------|            |           |----------|
|         |   |------------|   |   |          |
| Printer |___|    CPU     |___|   Disk       |
|         |   |            |       |          |
|---------|   |------------|       |----------|
              |      |     |
              |      |     |
              |------------|
              |  Terminal  |
              |            |
              |------------|
```

Figure 10. AUI Components

The addressable memory of the terminal will be in the range of 80K to 120K words. This maximum number is based on the transition from large scale integration (LSI) to very large scale integration (VLSI) and the introduction of multi-state technology to the central processing unit. The cpu processing speed should approach 200+ nanoseconds per instruction. The cpu should be a 16 bit word architecture with memory management, relocation, and memory protection. The processor will support floating point processing, stack processing, and direct memory access.

The terminal will also use disk storage devices for on-line and bulk storage of data and provide virtual memory and program swapping capabilities. The disk system will be a floppy or cartridge configuration with a capacity of one-half to two million words per unit. Average access time for the disks will be 50 milliseconds with data rates of 10 microseconds per word.

The CRT will support alphanumeric and graphic displays simultaneously in separate zones on the screen. The terminal will also provide highlighting features such as reverse video and blinking. A printer with the ability to reproduce the graphic displays will be required for hard copy output. The speed of the printer will be approximately 100 characters per second.

The terminal will interface to either syncronous or asyncronous communication lines. Control tables will provide flexible handling of special characters and protocols. The terminal will adaptively support whatever protocol is required to interface the terminal to the host computer.

2.4 AUI Overview

A breakdown of the functional components of the AUI is given below.

```
                          User
                           |
                           |
                           |
                           V    Screen Formatting
                          UIO   (User I/O)
                          DTF   Graphics and Lang.
                           |    Transformation
                           |    (Display
      |------|             |     Transfo
      | Check|             V     Function)
      | Help |        Mode Driver
      | List |             |
      |------|             |
                           V
      -------------------------------
      |          |         |          |
      V          V         V          V
     NDM       Record    Admin      Data
   scanning    level     Mode    Manipulation
     mode      query      |         Mode
      |        mode       V          |
      V          |      |---|        |
    |---|        V      |NDM|        V
    |NDM|     |-----|   |---|  |---------------|
    |---|     |AGENT|          |Working storage|
              |-----|          |---------------|
```

Figure 11. AUI Functional Components

The casual user asks questions from an intelligent terminal in the simplest manner. In reality, at a certain point, the user chooses among a limited number of possible choices -- each of which has the meaning of a question from the user. At any point, the HELP question is always acceptable, and the AUI details to the user what to do. Among the major different functions of the AUI, the NDM scanning mode, the data manipulation mode, and the virtual linkage mode are described as follows.

2.4.1 <u>Network Scanning Mode</u> The Network Data Model (NDM) fulfills the function of Data Dictionary, Data Directory, and Schema for the NVDB. Scanning the nodes of the NDM, the AUI processor will provide NDM-information to the user, and allow the user to request information whose existence was only indicated by the NDM, by switching to Record Level Query Mode (RLQM). Such detailed information is available to the NVDBS (AGENT) which, on request of the AUI, will extract it from its DBMS components, or rather their interfaces: the DMSREPs.

Typically, in the NDM scanning mode, the user will list the different sets available (LS submode), the different record types (LRT), the information associated with a specific set (SSN), or a specific record type (SRT). In particular, the user may want to list all sets in which a specific record type is involved, together with eventual membership options for these sets which are not owned by the record type.

At one point the user switches to RLQM and retrieves a particular record occurrence or perhaps all the member occurrences in the particular set occurrence of a set that this record owns. Further, he can "move the cursor" to anyone of the records listed, and again have a similar set of choices. An option to save or store the actual contents of records which are scanned is also constantly available. The following figure illustrates only the NDM scanning mode sequence.

```
Login under UNIX/AUI; get "screen mode";
these are your options:

|------| |----| |-------------| |------| |-----|
|Logoff| |NDM | |Record Level | |Data  | |VS & |
| AUI  | |Scan| |Query Mode   | |Manip.| |Admin|
|& HELP| |Mode| |(NVDBS Verbs)| |Mode  | |Mode |
|------| |----| |-------------| |------| |-----|
            |
            V
   - NDM SCANNING MODE -
```

Figure 12. NDM Scanning Mode

```
 Within the NDM Scanning Mode, one can select:
|------| |----| |------| |------| |------| |----|
|Logoff| |LS  | |LRT   | |SSN   | |SRT   | |Back|
| AUI  | |List| |List  | |Single| |Single| |Up  |
|  &   | |Sets| |Record| |Set   | |Record| |    |
|HELP  | |Mode| |Types | |Mode  | |Type  | |    |
|------| |----| |Mode  | |------| |Mode  | |----|
           |    |------|          |------|
           |
           V
       LS Submode
```

Figure 13. NDM Scanning Mode to LS Submode

```
Within the LS Submode, one can select:

|----||----||----||-----||-----||----||-----|
|Off ||Next||Prev||Selct||Selct||Back||Rec  |
|AUI ||page||page||singl||singl|| up ||Level|
| &  ||of  ||of  ||set  ||rec  || to ||Query|
|HELP||set ||set ||(SSN)||name  ||NDM ||Mode |
|----||list||list||-----||(SRN)||Scan||RLQM |
      |----||----|   |   |-----||Mode||-----|
                     |      |   |----|   |
                     V      |            |
                    SSN     V            V
                           SRN         RLQM
```

Figure 14. LS Submode options

Within the LRT Submode, one can select:

```
|----| |----| |----| |-----| |----| |-----|
|Off | |Next| |Prev| |Slect| |Back| |Rec  |
|AUI | |page| |page| |singl| | up | |Level|
| &  | | of | | of | |rec  | | to | |Query|
|HELP| |rec | |rec | |name | |NDM | |Mode |
|----| |list| |list| |-----| |Scan| |-----|
       |----| |----|    |    |Mode|    |
                        |    |----|    |
                        V              V
                       SRN           RLQM
```

Figure 15. LRT Submode and options

Within the SSN Submode, the display shows
Set name, Set owner, Set members
and the following options:

```
|----| |-----| |-----| |--------| |-----| |-----|
|Off | |Slect| |List | |List    | |Back | |Rec  |
|AUI | |rec  | |other| |other   | |up to| |level|
| &  | |type | |sets | |sets    | |NDM  | |Query|
|HELP| |in   | |owned| |where   | |Scan | |Mode |
|----| |set  | | by  | |member  | |Mode | |-----|
       |-----| |owner| |is      | |-----|
               |-----| |involved|
                       |--------|
```

Figure 16. SSN Submode and options

Within the SRT Submode, the display shows
Record name, Field names, Keys
and the following options:

```
|----| |------| |------| |------| |----| |-----|
|Off | |Rec   | |List  | |List  | |Back| |Rec  |
|AUI | |Access| |sets  | |sets  | |up  | |Level|
| &  | |Info  | |where | |where | |to  | |Query|
|HELP| |      | |rec   | |rec   | |NDM | |Mode |
|----| |------| |is    | | is   | |Scan| |-----|
                |owner | |member| |Mode|
                |------| |------| |----|
```

Figure 17. SRT Submode & Options

2.4.2 <u>The Data Manipulation Mode</u> The data which was extracted from the network and saved in working storage, can be manipulated by the user using text editing as well as a standard set of relational operations and aggregations, under the Data Manipulation Mode: the Tabular Algebra Language (TAL) and the Tabular Baby Language (TBL), which is translated into TAL.

When update is to be effected, the user may go back to the Network Data Scanning Mode. Placing the record cursor in the proper place, he will switch to the Record Level Query Mode, and retrieve the relevant record occurrence into working storage. The user will then switch to the Data Manipulation Mode, and update the copy of the record. The user will switch back to RLQM, and rewrite the record occurrence into the NVDBS. In turn, the NVDBS will trigger the proper update of the relevant DBMS of the network.

Note here that the Data Manipulation Mode is also meant to cover several functions which belong to the "NVDBS administrator" or the sophisticated user. These utilities can be operated through the TAS system used to manage the NDM as well as provide the data manipulation function under the AUI.

2.4.3 <u>The Virtual Linkage and Administrative Mode</u> The NDM scanning mode provides the user with the ability to navigate through the NDM. This scanning mode can be used by the analyst to become familiar with the sets and record types of the network and their relationships. Under the Virtual Linkage Mode, the NVDBS supports user-constructed personal virtual sets, in the following manner. The analyst would define a personal virtual set. After limited usage of this virtual set -- limited because this VS is not yet part of the NDM -- it is possible to have the VS fully integrated into the NDM. Limited usage means this: any NVDBS action such as deletion, which would normally propagate through the VS in question, would not propagate through it until it is made a full fledge "NVDBS" VS. It is only then that the VS is available to all users of the network.

At the beginning of a session, a user or a user-process will trigger an initialization phase, performed under TAS. During that phase, the personal VSs of the user -- which were kept on the local AUI host -- are loaded into the NDM with the user's tag instead of the NVDBS tag. Normal NVDBS VSs are tagged "NVDBS". When the AGENT triggers certain transactions which propagate through different DBMS on the network, the NVDBS will check to see if the VS references are "normal" or "personal". "Personal" VS references will be ignored at the network level unless the VS tag matches that of the user. In this way the user may change and explore virtual sets on the "personal" level without fear of destroying or modifying network level virtual sets. This mechanism also protects the user's "personal" VSs from being used by others and protects the NVDBS from having "personal" VSs affect the network virtual sets. In the next section, NVDBS verbs which are used by both the AUI and the AGENT are reviewed. They essentially cover the "RLQ" mode. The final section gives an example of AUI modes and usage.

2.5 AUI, AGENT verbs, and Record Level Query Mode

Most of the following verbs are used by the AUI at the Record Level Query Mode and by the AGENT in its dealings with the DMSREPs.

2.5.1 LOCATE: The format is

LOCATE {record-name} {WITHIN set-name} {USING identifier} {WHERE clause}

LOCATE alone will return the DBK for a record occurrence of the current record in the current set occurrence. If there is no current set occurrence but there is a current set name the first set occurrence is selected. If there is no current set name or if there is no current record, an error message is returned.

LOCATE with the specification of "record name", will first reassign the current record name, then will act as above. The current record name must be a member or owner of the current set name, or else an error condition is returned.

LOCATE ... WITHIN set name, will reassign the current set name. The current record name must belong to the current set name.

LOCATE ... USING identifier, will have the constant value in ASCII ultimately translated into the correct alphabetical or numeric specification for ultimate use by the DMSREP in querying the relevant DBMS.

LOCATE ... WHERE clause, specifies a condition (simple monadic predicate) to be verified by the key value of a record occurrence.

After a LOCATE, the DBK of the current record is reset, and buffers and currencies are ready for a READ.

This command is similar to the CODASYL "FIND record type WHERE clause" or the [S2000/74] "LOCATE" command.

2.5.2 <u>MOVE CURSOR</u>: The format is

```
MOVE_CURSOR  NEXT
             PRIOR
             FIRST
             LAST
             OWNER
             MEMBER member name
```

will "LOCATE" the NEXT (or PRIOR/FIRST/LAST/OWNER) record occurrence for the current record. The NEXT option is always valid. All other options are only valid if the current set allows for them. Information may be available from the NDM. Surprises may be avoided by using the PUSH_CURSOR and POP_CURSOR commands, to save the currency and DBK of previous occurrences of set owner and member(s). A special option to push automatically before every change of currency may be activated.

The OWNER option may return a "no owner" error message. The MEMBER option returns an error when the member-name is not a member of the current set name. If MEMBER option is used, and no member name is specified, the NVDBS will give the user the choice between the different member names of the current set name, if there are more than one. If there is only one, the system will automatically provide the name.

Most DBMSs have consistent facilities. IMS and S2000 always have the NEXT option in a similar command. CODASYL types have the following command:

```
FIND |-|-NEXT--|--record name-| RECORD OF
     | |-PRIOR-|              |  set name
     | |-FIRST-|              |      SET
     | |-LAST--|              |
     |                        |
     |---OWNER----------------|
```

The record occurrence is retrieved, but is not copied to working storage. If GET (or OBTAIN) are used, instead of find, the occurrence is also moved to working storage.

2.5.3 <u>POP CURSOR</u>/<u>PUSH CURSOR</u>: These commands require more action from the NVDBS itself than from the constituent DBMSs. The only effect on the DBMS is in terms of CURRENCY. A stack structure exists in the AGENT working storage. After a POP_CURSOR, the new CURRENT record occurrence is clobbered by the top of the stack record occurrence, which may be of a different type. After a PUSH_CURSOR, nothing in the DBMS is changed, and the current record and occurrence is all stored on top of the stack. In the case of a CODASYL DBTG type of DBMS, the MOVE CURRENCY STATUS would be used by the DMSREP in order to execute the PUSH_CURSOR.

2.5.4 <u>READ</u>: The READ command gets the record occurrence out of the DBMS into the DMSREP and the AGENT working storage. The first phase of the READ command maps exactly into a CODASYL GET command. The second phase sees the DMSREP send the record occurrence to the AGENT. If the AGENT transaction is completed, the AGENT will then forward it to the AUI. An NVDBS READ is therefore somewhat equivalent to a CODASYL "GET record name RECORD", followed by a DISPLAY or WRITE command (PRINT command in S2000).

2.5.5 <u>MODIFY</u>: This command maps into the CODASYL command (*) to modify a particular record occurrence. The format is:

```
MODIFY record name RECORD
```

The current record occurrence must be of the "record name" type. This occurrence is then modified by a corresponding record layout in working storage.

(*) The corresponding S2000 command is: "CHANGE data element". Note S2000 also uses ASSIGN, which assigns data within data sets whether or not the existing data sets contain data.

2.5.6 <u>DELETE</u>/<u>STORE</u>/<u>CONNECT</u>/<u>DISCONNECT</u>: These are commands relative to the current record and set occurrences. They are detailed in section 3.

2.5.7 <u>STATUS INQUIRY</u>: The purpose of this command is to probe at different levels of the NVDBS. From the user's console, a STATUS_INQUIRY will display all currency indicators and the state of the AGENT. The AGENT addresses a STATUS_INQUIRY to any DMSREP to find out the state of the particular DMSREP/DBMS interface.

2.6 VERBS FOR AGENT ONLY

2.6.1 <u>BEGIN TRANS</u>/<u>END TRANS</u>: Although these verbs are under the "Agent Only" heading, there is the need to have their implicit meaning present at the AUI level. The particular manner in which this will be carried out will depend upon how the user will be able to batch up a series of AUI commands. Note also, on this topic, that the addition of a "HOLD" verb similar to [SYSTEM R] is under consideration.

These verbs delimit a "transaction", i.e. a sequence of commands which should be executed in one block. A module of the network sends a transaction to another module and this other module will wait for the END_TRANS and consider processing the transaction only at that point. Within a transaction, certain commands may require certain accesses and locks will be set accordingly at execution time. The meaning of the transaction includes the fact that the locks are not relinquished untile the end of the transaction's execution. Consistency is at stake here, and the transaction is the vehicle to execute certain updates while preserving the database consistency.

Transactions may be nested if the innermost transaction is in fact to be forwarded to another module. The commands do not necessarily get mapped into DBMS verbs, but there are some close similarities. For example, in [S2000] "queue processing", the set of commands delimited by a beginning QUEUE command and terminating TERMINATE command are processed as a unit similar to the BEGIN_TRANS/END_TRANS.

2.6.2 <u>INVOKE</u>/<u>ENDINV</u>: The AGENT queries the NDM and finds out that it must INVOKE a particular subschema in a particular DBMS. The NVDBS INVOKE command triggers in the DMSREP a correspondent CODASYL type INVOKE followed by an OPEN relative to the appropriate area, if we assume that the DBMS is a CODASYL type.

ENDINV represents the end of the particular session with the DMSREP. Such a command should be typically followed by an END_TRANS.

In the case of a CODASYL DBMS, the DMSREP must issue the following command:

```
INVOKE SUBSCHEMA subschemaname OF schemaname
OPEN|area USAGE MODE IS|RETRIEVAL----------|
    |ALL AREAS         |PROTECTED RETRIEVAL|
                       |EXCLUSIVE RETRIEVAL|
                       |UPDATE-------------|
                       |PROTECTED UPDATE---|
                       |EXCLUSIVE UPDATE---|
```

Therefore, the appropriate arguments must be found by the AGENT and passed to the DMSREP. In [S2000], the similar commands are LOAD/UNLOAD.

2.7 <u>VERBS FOR AUI ADMINISTRATIVE MODE</u>

2.7.1 <u>CREATEVS</u>/<u>DESTROYVS</u>: These commands create/destroy virtual sets [NVDBS78/1] in the Network Data Model. Their format is

```
CREATEVS set name, owner name,
  member name #1, option for #1,
  member name #2, option for #2,
  ... etc...
DESTROYVS set name
```

After a CREATEVS, an entry is created in the VST table, and either a VSOT must be created or a corresponding recipe must be stored, which could have been created through the appropriate Data Manipulation TAS facility. After a DESTROYVS, the VST entry and the VSOT or "VS-recipe" are erased. Of course, both of these commands would be complemented with an assortment of prompters to extract auxiliary information from the user.

2.7.2 DESCRIBE: This command is sent by the NVDBS for the purpose of obtaining from the DMSREP the data base description. This may be to update the NDM, after a DBMS reorganization, or schema update.

2.8 EXAMPLE OF AUI USAGE IN THE DIFFERENT MODES

The following example is given to illustrate the ease of use of the AUI and the functions of the verbs which were briefly described above. This example assumes the user has a touch panel entry terminal. Consider the following sample data, against which the user will exercise his skills under the Adaptive User Interface:

Set Name	Owner	Member
Out-Traffic	Airport	Flight
Crew_Complement	Aircraft	Specialty
Domicile	Airport	Personnel
Crew_Assignment	Aircraft	Personnel
Equipment_Assignment	Aircraft	Flight
Flight_Crew	Flight	Personnel

Figure 18. Airline Example Data; Information Structure

```
|----------------------------------------------------|
|Aircraft:                                           |
|   |------------------|--------------|---------|    |
|   |Airline/Aircraft# |Aircraft_type |Capacity |    |
|   |------------------|--------------|---------|    |
|    UA/130:B727:122                                 |
|    AA/431:B707:184                                 |
|    TWA/29:B747:291                                 |
|Airport:                                            |
|   |---------|-------------|-----|                  |
|   |Mnemonic |Airport_name |City |                  |
|   |---------|-------------|-----|                  |
|    SFO:San Francisco International:San Francisco   |
|    LAX:Los Angeles International:Los Angeles       |
|    BOS:Logan International:Boston                  |
|    PHX:Sky Harbor:Phoenix                          |
|    IAD:Dulles International:Washington D.C.        |
|----------------------------------------------------|
```

Figure 19. Airline Example Data: Actual Record Occurrences

```
Personnel:
  |-----|-----------|------------------|
  |Name |Employee#  |Years_Experience  |
  |-----|-----------|------------------|
   Elise Granville:11239:7
   Scarlet O'Hara:67349:9
   Maragaret Mid:84902:12
   Evil Roy Slade:44439:4
   Tom Mix:52301:6
Specialty:
  |-----|------|
  |Type |Title |
  |-----|------|
   Tech:Captain
   Tech:First_Officier
   Tech:Flight_Engr
   Serv:In_Flight_Supvr
   Serv:Flight_Attendent:B747
Flight:
|-----------------|----------|-------------|-------|
|Airline/Flight#  |#1st_Pass |#Coach_Pass  |Origin |
|-----------------|----------|-------|-----|-------|
     86:14:122:SFO:BOS                |Destination |
     141:22:230:LAX:PHX               |------------|
     833:38:382:SFO:IAD
     690:6:221:BOS:PHX
```

Figure 20. Airline Example Data: Record Occurrences

The user may first wish to scan the NDM to see what data is available (record types) and the relationships - called sets - between record types. The NDM Scanning Mode gives the following schema display for the airline sample data (for network sized data models, the user would specify a starting record type or set type in order to localize the display):

```
|--------|---->Crew_Assignment-----------|
|Aircraft|-->Equipment_Assignment        |
|--------|              |                |
   |                    V                |
   |                 |------|            |
   |      --------->|Flight|->Flight_Crew |
   |      |          |------|        |   |
   |    Out_Traffic                  |   |
   |      ^                          |   |
   |      |                          V   V
   |   |-------|                  |---------|
   |   |Airport|----> Domicile---->|Personnel|
   |   |-------|                  |----^----|
   |                                   |
   |                 |---------|       |
   v  Crew_    ---->|Specialty|--->   Crew_
    Complement      |---------|     Assignment
```

Figure 21. Airline Data Model

The user could also list the sets or record types. From either the lists or the data model display the user may point to a partiicular set or record type. Pointing to the Personnel record type will display the following record structure information on the terminal (the starred field indicates a data base key field):

```
          Specific Record Type Mode

         Record Type Name:  Personnel

     Field Identifier   Field Length  Field Type
   |-------------------|-------------|----------|
   |Employee_name      |     20      |  Chars   |
   |-------------------|-------------|----------|
  *|Employee_number    |      5      |  Integ   |
   |-------------------|-------------|----------|
   |Seniority          |      2      |  Integ   |
   |-------------------|-------------|----------|

  |----| |------| |------| |------| |----| |------|
  |Off | |Record| |List  | |List  | |Back| |Record|
  |AUI | |Access| |sets  | |sets  | | up | |Level |
  | &  | |Info  | |where | |where | | to | |Query |
  |HELP| |------| |record| |record| |NDM | |Mode  |
  |----|          | is   | | is   | |Scan| |------|
                  |owner | |member| |Mode|
                  |------| |------| |----|
```
Figure 22. (Personnel) Record Type Format Display

Once a specific record type is selected the user may select several different displays to learn more about that record type. The user can, for example, ask for all the sets in which the Personnel record type is a member. This information will tell the user the different grouping that personnel can be divided into or associated with. In the Personnel record example, the Personnel record is used in the Domicile, Flight_ Crew, and Crew_Assignment sets. The user, upon his request, would also find that the Personnel record is not the owner record of any sets. A request for the list of sets of which the Airport record type is owner would tell the user that for each Airport record there is both Domicile and Out-traffic information. If the user wanted to know the structure of the Domicile set he need only point to it and the display would be presented:

```
Specific Set Name Mode

Set Name:  Domicile
|-------------|-------------|-------------------|
|Owner Record |Member Record|Membership Options |
|-------------|-------------|-------------------|
|Airport      |Personnel    |Automatic-Mandatory|
|-------------|-------------|-------------------|

|------|  |-----------|  |-----------|  |----|  |------|
|Logoff|  |List other |  |List other |  |NDM |  |Record|
| AUI  |  |sets where |  |sets where |  |Mode|  |Level |
|      |  |record =   |  |record =   |  |    |  |Query |
|& HELP|  |owner      |  |member     |  |    |  |Mode  |
|------|  |-----------|  |-----------|  |----|  |------|
```

Figure 23. (Domicile) Set Description

This display will tell the user that for each airport there is a set of personnel records for the airline personnel that live near or work from that airport. This display also tells the user that Personnel record occurrences that are added to the data base will automatically be added to the Domicile set. The mandatory means that once in a Domicile set occurrence, the Personnel record occurrence can never be removed from the set occurrence unless the person quits and the Personnel record is deleted or the person moves to a new domicile set occurrence. Touching the fields "Personnel" or "Airport" (member or owner) will display the Specific Record Type requested.

When the user wants to look at actual record occurrences, he touches the mode button labeled Record Level Query Mode. The RLQM requires that the set and record type be specified. The RLQM will use the last set and record type specified in the NDM Scanning Mode or the user may specify the set and record type desired. The RLQM displays the set and record type information along with the command buttons to perform record level retrievals. The RLQM format is:

```
              Record Level Query Mode
Set Name: Domicile                                          |------|
 Owner    Member      SMO                                   |LOCATE|
 Airport  Personnel   Auto/Mand                             |------|
                                                       |----| NEXT |
Owner Record: Airport                                  |    |------|
   *Mnemonic           |_|_|_|                         |    |PRIOR |
    Airport_name                                   Cursor   |------|
    City                                               |    |FIRST |
                                                       |    |------|
Member Record: Personnel                               |----| LAST |
    Employee_name                                           |------|
   *Employee_number        |_|_|_|_|                        | READ |
    Seniority                                               |------|
                                                            |STATUS|
                                                            |------|
                                                            |DELETE|
|----| |----| |------| |-------| |--------|                 |------|
|Off | |NDM | |Data  | |Virtual| |Remember|                 |STORE |
|AUI/| |Scan| |Manip.| |Link   | |--------|                 |------|
|HELP| |Mode| |Mode  | |Mode   | |Execute |                 |MODIFY|
|----| |----| |------| |-------| |--------|                 |------|
```

Figure 24. Example of Record Level Query Mode

The user may locate the domicile of a particular employee, whose employee number he knows, by touching Employee_number, typing 67349 in the spaces next to Employee_number, and then touching the LOCATE button. Once the employee record occurrence of Scarlet O'Hara is located, the user touches Airport and the READ button to obtain the domicile information of Scarlet. You then learn that Scarlet O'Hara is based in San Francisco.

This sequence of commands could be executed one at a time as the screen is touched or they could be REMEMBERed and then executed as a group by touching the EXECUTE button. REMEMBERed sequences can also be stored for later execution in the embedded batch mode.

If the user touched "Personnel" and the READ button, the personnel record of Scarlet O'Hara would be read into AUI working storage. Touching NEXT and

READ in sequence twice would read the personnel records of Margaret Mid and Evil Roy Slade. The user could then change to the Data Manipulation Mode to display and manipulate the records in AUI working storage. Using relational specifications the user could ask for a display of the name and seniority of all personnel with more than ten years seniority. This would display Margaret Mid and 12 years. (The mechanisms associated with the data manipulation mode, i.e. the REalational System, is the object of another paper [NVDBS78/4].) The user could return to the NDM Scanning Mode to find what other information was available about Margaret Mid. He would see that Flight_Crew and Crew_Assignment information (set) is available for Personnel record types. The RLQM could be used to retrieve this information into working storage and the Data Manipulation Mode could then be used to construct a relation with all the information on Margaret Mid including her domicile, flight, and specialty. All of this information might be required by a scheduler in assigning Margaret to a Flight. The scheduler would first identify the Airport and Out-traffic Flights to be scheduled. From the list of Flights he would determine (using the Flight_Crew set: "Flights" where owner is "Flight" and member is "Personnel") which flight did not have a full Flight_Crew and what type of Aircraft was to be used for the Flight. The scheduler already knows that Margaret's Specialty is captain. By following the Crew_Complement set he could also determine that she was certified to fly a Boeing 747. Once he has retrieved all of the Flights with partial crews, he would, in the Data Manipulation Mode, query for all flights using a 747 and requiring a captain.

The approach, illustrated by the above example, simplifies the manipulation of set and record types, and their occurrences.

2.9 CONCLUSIONS ON SECTION 2

The main functions of the AUI have been reviewed. In particular, the NDM scanning mode, and the RLQ mode were discussed in some details (both the Data Manipulation mode and the Virtual Sets and Administrative mode are explained in later sections). A basic set of network record level commands was presented. Such commands allow record-at-a-time operations by the network on the data base components, within the set structure. The set structure, through its extension: the virtual set, provides the vehicle for the system to automatically access the different DBMSs with complete transparency.

3. RETRIEVAL AND UPDATE VERBS AND SET MEMBERSHIP OPTIONS

3.1 RETRIEVAL AND UPDATE FUNCTIONS

There are essentially three interface levels in the NVDBS.

3.1.1 The User/AUI level: The user exercise the choices or uses the language provided by the Adaptive User Interface (AUI). There are four modes of interaction for the user:

1. The NDM scanning mode,
2. The Data Manipulation mode,
3. The Record Level Query mode (RLQM)
4. The Virtual Set and Administrative mode

The NDM scanning mode is for AUI retrieval access to the NDM. The NDM scanning mode is mostly graphic and does not use hardly any language. The Data Manipulation mode uses a tabular system (TAS) which is not very different from most relational systems [SYSTEM R][INGRES]. It accesses local data in working storage. The RLQM is the mode for AUI to access the different DBMSs on the network. The RLQM uses a set of verbs, some of which will be discussed below. The Virtual Set and Administrative mode accesses and updates the NDM and the AGENT through RES and canned utilities.

3.1.2 The AUI-to-NDM-and-AGENT level: The AUI retrieves information from the NDM (for instance, in the NDM scanning mode, or in the Administrative mode). TAS is used to access the NDM. The AUI passes to the AGENT these RLQM commands discussed below.

3.1.3 <u>The AGENT-to-NDM-and-REPs level</u>: The AGENT answers the RLQM queries from the AUI. The AGENT accesses the NDM through RES. The AGENT generates some transactions which are mailed to the REPs. In some instances, an RLQM command triggers complex AGENT transactions, e.g. a "DELETE SELECTIVE" issued by the AUI to the AGENT may trigger many different "DELETE SELECTIVE" and "DISCONNECT" commands, when:

A. at least one DBMS does not support the CODASYL Set Membership Options (SMO), and consequently these SMOs are the responsibility of the NVDBS,

B. a Virtual Set (VS) is involved [NVDBS78] and the "DELETE SELECTIVE" must be propagated across different DBMSs.

Because the NVDBS is only a user with respect to the DBMSs on the network, it is sometimes essential that the AGENT finds out ahead of time whether or not all the intended actions resulting from an AUI query will be authorized by the different DBMSs involved. This exploratory phase includes the analysis of AUI request, the generation of intended AGENT commands, and the verification of acceptability by relevant DBMSs through exploratory transactions. Much is expected from the DMSREPs on the subject of authorizations, for it is not always possible to experiment in order to discover access authorization. Indeed because authorizations levels often look like a partial ordering, or a lattice, there is no guarantee that the reverse of an update action be authorized. Situations where some data bases would have been submitted to aborted updates could lead to inconsistencies and meaningless data. Section 5 will go in more details on some of these considerations. They are related to these verbs which perform updates. In conclusion, the exploratory phase is the price for some consistency in the overall NVDBS and for simpler recovery mechanisms.

3.2 SET MEMBERSHIP OPTIONS AND RELATED RLQM COMMANDS

Consider the deletion, addition, removal from a set-membership, and acquisition of set membership for a given (current) record occurrence. The verbs are:

DELETE (alone/ONLY/SELECTIVE/ALL)
STORE
DISCONNECT (or remove)
CONNECT (or insert)

and before going into further explanations, let us review the Set Membership Options (SMOs) in a CODASYL-like system.

3.2.1 Set Membership Options

SMOs are presented in [CODASYL-DBTG71]. They are carried in the Real Sets Table (RST) and Virtual Sets Table (VST) in the NDM. Before going into any further detail, recall that:

- STORE will store a record-type occurrence from working storage to the data base.
- DELETE will delete a record-type occurrence from the data base depending upon certain options.
- CONNECT will establish the membership of a record-type occurrence into a specified set. This works provided the membership is either OPTIONAL or MANUAL. For instance, a record type occurrence is store. The MANUAL membership set requires a STORE followed by a CONNECT in order to insure that the record type occurrence becomes a member.

- DISCONNECT removes a particular record-type occurrence from set membership in the current set occurrence.

In order to describe more specifically the four above verbs one should explain that set membership must be either MANDATORY or OPTIONAL, and that it also must be either AUTOMATIC or MANUAL.

3.2.1.1 MANDATORY/OPTIONAL

This option relates to the DISCONNECTion or DELETion of old record occurrences. MANDATORY membership means that a member record occurrence cannot be DISCONNECTed from the membership set, once the membership has been established; but it can be moved (MODIFY) from one set occurrence to another. OPTIONAL membership, on the contrary, means that a member record occurrence may be DISCONNECTed from the membership set to which it belongs.

3.2.1.2 AUTOMATIC/MANUAL

This option is relative to storage of new record occurrences, or insertion of existing record occurrences into existing membership sets. An AUTOMATIC membership option for a given set name means that any record occurrence which is STOREd into the data base, and for which the record name is an AUTOMATIC member of some set, will be AUTOMATICally CONNECTed into the membership set under the current owner. A MANUAL membership will mean, on the contrary, that the STORE must be followed by an CONNECT so that the particular record occurrence which was STOREd become in fact a member occurrence in the current set occurrence.

3.2.1.3 The different DELETES

There are four kinds of DELETE options.

1. DELETE alone applied to a record occurrence DELETEs such a record occurrence[deletes such record occurrence only if it is not the owner of any non empty set occurrence.

2. DELETE ONLY applied to an owner record occurrence will delete all MANDATORY members occurrences of the set occurrence whose owner occurrence which is being deleted. On the other hand, the same DELETE ONLY will not delete the OPTIONAL membership record occurrences which were owned by the owner_record occurrence under deletion: it will only remove them.

3. DELETE SELECTIVE, like all other DELETEs, deletes the current record occurrence and beyond this, it also deletes all MANDATORY members occurrences in the owned sets occurrences, if any. But DELETE SELECTIVE will only DISCONNECT these OPTIONAL members occurrences which are involved in some other sets occurrences, and here, other sets occurrences mean these which are not owned by the current record occurrence.

3.2.2 Superposition of NVDBS and DBMS SET-MEMBERSHIP-OPTIONS

Whether or not a particular RLQM command implicates a virtual set, there is a need for the NVDBS to do some housekeeping before and after it passes the corresponding commands to the relevant DBMS. Indeed, the NVDBS must translate a transaction (originating from the AUI and targeted for the virtual view of integrated data) into a set of actions, some of which are at the NDM level and others which are transactions with the different DMSREPs and their associated DBMSs.

Because of the necessity for the NVDBS to "integrate" different DBMSs including the CODASYL type, it results that the different set membership options must be implemented at the network level. Indeed, if these options did not exist at the network level, an expensive dialogue between a necessarily more complex DMSREP and NVDBS_AGENT would take place. This would also require that the NDM maintain an heterogeneous view of the data.

As it is, with the options at the network level, it is relatively easy for the NVDBS to maintain coordination between the effects of SMOs at the network level and the effects of SMOs at some DBMS level. Last but not least, we note that the DBMSs which do not have SMOs at the DBMS level (non-CODASYL) still have virtual SMOs at the NVDBS level. This does not conflict with typical DBMSs.

In summary, if an underlying DBMS has a set with certain SMOs, the same SMOs must exist at the network level (in the NDM and in the NST tables). If an underlying DBMS does not carry the CODASYL SMOs, the NST table in the NDM will carry the information, and the NVDBS will enforce these SMOs. In all cases, the effects of these SMOs will be propagated across the DBMS boundaries through the virtual sets and their SMOs (which clearly exist only at the network level).

The SMOs were reviewed in the previous section. They are:

MANDATORY/OPTIONAL

(effect on DELETE and DISCONNECT)

AUTOMATIC/MANUAL

(effect on STORE and CONNECT)

The relevant commands are:

DELETE alone "R"

DELETE ONLY "R"

DELETE SELECTIVE "R"

DELETE ALL "R"

STORE "R"

CONNECT "R"

DISCONNECT "R".

Typically, these commands originate from the AUI and pertain to the integrated view maintained in the NDM. We shall briefly go over the AGENT's algorithms to handle these commands.

The mechanisms described below are triggered whenever a STORE, CONNECT, DELETE, or DISCONNECT command is issued from the AUI to the NVDBS_DMSREP. The assumptions are that the NVDBS maintains current "pointers" or "cursors" or "data base keys" for all record types and sets in the AUI window, in a typical manner. Currency is what defines implicitly the arguments for the verbs above.

3.2.2.1 <u>The strategy</u>:

Because of the complexity of the mechanisms involved, an exploratory phase will precede the execution phase of any update action directed by the AUI to the AGENT. The exploratory phase will check as thoroughly as possible that all actions required will in fact be permitted by the real DBMSs on the network. If during the propagation of a certain command (e.g. a DELETE) there occurs the exercise of a restriction placed by the real DBMS on the DMSREP, then the whole AUI transaction will be shelved. The NVDBS will then inform the AUI of where the restriction occurred and, if possible, why. The AUI may then inform the user of the problem and suggest alternative actions.

The mechanisms assume that the real DBMS on the host node is a CODASYL type, with the SMOs incorporated. If this is not the case, then the NVDBS takes advantage of the network level SMOs indicated in the NDM: specifically in the Real Set Table (RST). The AGENT outputs the correct commands to the non-CODASYL DBMS so as to make it act globally like a CODASYL system, thereby guaranteeing a consistent SMO implementation for the virtual data base.

3.2.2.2 Details on DELETE/STORE/CONNECT/DISCONNECT

3.2.2.2.1 DELETE (alone) 'R'

1. The NVDBS checks current record occurrence is of type "R"; else error (currency).

2. The NVDBS checks that there is no virtual or real set where "R" is owner. If there is a virtual set where "R" is owner, the NVDBS checks that there is no non-empty set occurrence owned by the "R" occurrence; else error.

3. The NVDBS looks for all virtual sets where "R" is member, and accesses the corresponding VSOT tables, in order to delete the "R" occurrence key from any set occurrence (VSOT entry) where it might appear.

4. The NVDBS sends the order to the DBMS in charge of "R" to delete the "R" occurrence which is current. This action must be "explored" prior to the actual execution of DELETE (alone).

Note that the NVDBS must not actually execute any deletion step before the exploratory phase has determined that all steps were executable.

3.2.2.2.2 <u>DELETE ONLY 'R'</u>

1. The NVDBS checks that current record occurrence is "R" type; else currency error.

2. All virtual sets where "R" is owner and all set occurrences where "R" occurrence is owner must be found. For such occurrences, all MANDATORY member occurrences must be the object of DELETE ONLY commands issued by the NVDBS and sent to the relevant real DBMSs. For such set occurrences, all member occurrences - whether MANDATORY or OPTIONAL - are removed from the set occurrence which is itself destroyed, i.e. the VSOT entries corresponding to the set occurrences are erased. Other VSOT entries (containing the member occurrence as OPTIONAL member occurrence) are shortened. A listing of record occurrence keys of the MANDATORY members occurrences which will be deleted by the real DBMSs must be obtained, in the case when these MANDATORY members are involved in some virtual sets. The NVDBS must then propagate the "DELETE ONLY" to other real DBMSs if the virtual sets have mandatory members in other real DBMSs, and the occurrences are non-empty. Again, other VSOT entries (containing the member occurrences as OPTIONAL) must be shortened.

   A. Note that irreversible actions should not be taken by the NVDBS until the relevant real DBMSs have executed the commands sent by the NVDBS_AGENT. These must be explored first. All MANDATORY member occurrences must be satisfactorily subjected to the DELETE ONLY before the NVDBS proceeds to make network level modifications as indicated, i.e. removal of all member occurrences from VSOT entries.

B. Note that the NVDBS actions govern the actions taken by real DBMSs underneath. Yet the NVDBS acts like an ordinary user, e.g. if an NVDBS-required deletion is not authorized by the real DBMS underneath, then the whole transaction should be rejected at exploration time.

3. All virtual sets where "R" is a member and where a set occurrence (VSOT entry) contains the "R" occurrence, are formed. The corresponding VSOT entries are modified to reflect the deletion of the "R" occurrence.

Again, the NVDBS does not request any real deletion in any DBMS before the exploratory phase has returned a satisfactory answer, i.e. that all required deletions are executable. Consequently the exploratory phase includes an algorithm which follows the directed graph of information structure and computes the "explosion" of the different sets occurrences owned by the current record occurrence; this algorithm also checks that each deletion is authorized. If there is at least one required but unauthorized deletion, the algorithm stops its search and the AGENT returns a documented "cannot do" to the AUI.

3.2.3 <u>DELETE SELECTIVE 'R'</u>

1. The NVDBS checks that current record is type "R"; else currency error.

2. Steps similar to DELETE ONLY procedure must be taken, first the exploratory phase and then execution.

The differences between DELETE SELECTIVE and DELETE ONLY include the following:

- DELETE SELECTIVE may propagate along the sets owned by the OPTIONAL members occurrences. DELETE ONLY does not: it may propagate only through

MANDATORY members occurrences.

- Whenever a virtual set occurrence is involved, the OPTIONAL member occurrences are to be deleted if removal makes them orphans. The NVDBS issues a DELETE SELECTIVE on these OPTIONAL member occurrences only after checking (exploration) that they are not member occurrences in any other set (whether VS -- and this is easy to check in the VSOT -- or RS, which is more expensive because it involves an AGENT transaction towards the relevant DBMS to check the set occurrences). In the case when the OPTIONAL member type is not a member of any other set, the NVDBS can find this out by merely scanning the NST and VST, and decide to send a DELETE SELECTIVE immediately). The first time the AGENT finds that a particular OPTIONAL member occurrence is not going to be an orphan, it merely removes that one occurrence (VSOT entry is erased) and does not perform the explosion through this particular occurrence. If on the contrary, the particular OPTIONAL member occurrence is actually an orphan, then a "DELETE SELECTIVE" must be applied to it, and the whole exploratory procedure must be reentered recursively.

3.2.3.0.1 <u>DELETE ALL 'R'</u>

1. The NVDBS checks that current record is type "R"; else currency error.

2. The NVDBS scans the VST and VSOT and propagates the DELETE ALL order to all occurrences of members in the relevant set occurrence owned by the current "R" record occurrence.

3. The NVDBS also passes the DELETE ALL "R" (as is) to the real DBMS concerned, after exploration; and it propagates it eventually across virtual

set occurrences.

4. The NVDBS deletes the "R" occurrence from any VSOT entry where "R" occurrence is a member occurrence.

In [S2000] the DELETE ALL is equivalent to the REMOVE TREE where the cursor is pointing to the root.

3.2.3.0.2 <u>STORE</u> <u>'R'</u>

1. The NVDBS must check that the run-unit record (current record occurrence) is of the "R" type; else currency error.

2. The NVDBS scans the VST and looks for sets where "R" is an AUTOMATIC member. For each such set occurrence that it finds, it must then access the corresponding VSOT and look for the appropriate key-value of the owner for the set occurrence in which the "R" occurrence must be automatically inserted. Presumably, this key-value has been given by the AUI. The default assumption in [CODASYL-DBTG71] is that the correct key-value is the current key-value (current set occurrence).

   The entry in the VSOT which corresponds to the correct set occurrence is extended to include the new member occurrence ("R" occurrence). Simultaneously, the NVDBS issues a STORE "R" command to the DMSREP of the DBMS containing the record R and all of its occurrences. Also it scans the VST for sets where "R" is owner. It gets the corresponding VSOT and adds to it a new entry with the "R" occurrence key-value. The corresponding set occurrence is clearly empty.

3.2.3.0.3 CONNECT 'R'

1. The NVDBS checks the current record is of type "R"; else currency error.

2. CONNECT "R" means that the system must connect (insert) the current record occurrence (in run-unit) into a set occurrence characterized by the owner key (current owner occurrence).

   If the current set is a virtual set, then the whole connection (insertion) process takes place at the network level, in the corresponding VSOT table. If the current set is a RS (belonging to a specific DBMS) then the NVDBS passes the CONNECT command to the DBMS in question, through a transaction sent to the corresponding DMSREP.

3.2.3.0.4 DISCONNECT 'R'

1. The NVDBS checks that the current record occurrence is of type "R"; else currency error.

2. The NVDBS looks at current set_name. If current set is virtual, then the entire operation is virtual, and performed at the network level.

   A. The VST is accessed and "R" is checked to be indeed the member of the current virtual set, characterized by an entry in the VST.

   B. The NVDBS follows the pointer to the VSOT table and updates the VSOT entry corresponding to the current set occurrence (i.e. it suppresses the key field containing the key value for the "R" occurrence.

3.3 CONCLUSIONS

This section has presented the manner in which the NVDBS handles the SMOs of the owner-member set structure adopted in the system. In conclusion, it is interesting to consider the notion of consistency at the level of the NVDB. On one hand, our first principle [NVDBS78] is the "non-interference" of the NVDBS with the different DBMSs. On the other hand, the NVDBS is enforcing some SMO rules across the boundaries of the DBMSs, at the network level. Clearly, since the DBMSs function as stand alone, and remain "free" from the NVDBS, there seems to be little point in spending so much effort in enforcing the SMO rules at the network level. Indeed, each DBMS alone is free to take any action including some action which would violate these SMOs maintained at the NVDBS level. The answer to this is that we have merely proposed the first step towards a more complete integration. For instance, the next step may be that the DBMSs accomplish their updates through the NVDBS, after the fact. In our view, the whole merit of the NVDBS is to provide an approach which

- builds upon the work already accomplished in the DBMS field,
- remains very modular and progressive
- allows for variable integration of the DBMS-components of the NVDBS.

4. DATA MANIPULATION MODE FOR THE ADAPTIVE USER INTERFACE

The Data Manipulation Mode (DMM) is provided for the user to manipulate the data which was extrated from the network and stored in local storage. A typical example is illustrated in the following:

A user sits in front of a CRT terminal connected to an Adaptive User Interface (AUI). The user first starts in the Network Data Scanning Mode (NDSM) to examine the logical structure of the Network Virtual Data Base (NVDB), then switches to the Record Level Query Mode (RLQM) to retrieve the relevant record occurences from the network and stores them into the local storage of the AUI. The user will then switch to the Data Manipulation Mode (DMM), and manipulate copies of these record occurences. When update to the original network data base is to be effected, the user will switch back to RLQM and update the particular record occurences which are to be updated in the NVDB.

In this section, we first present an overview and the design goals of the DMM, and then a brief description of the functional organizations and the major components of the DMM. After having these general features of the DMM described, the distinctive features of the TAB and TAL languages are presented in detail. This includes the summaries of the commands and options, and the illustrative examples of each command and option.

4.1 OVERVIEW

As described in a previous section, the Adaptive User Interface (AUI) is designed to be a user friendly front-end to the NVDBS. Four interaction modes are provided by the AUI and each mode utilizes state-of-the-art display terminals, with either a touch panel or cursor addressable facility, to offer a simple "dialog" for the casual users. The simple display dialog offered by the Data Manipulation Mode (DMM) is called the TAB (TABular language). The TAB is a table data language similar to "Query-by-Example" as presented in [Zloof75/76/77]. In addition to the TAB dialog, the DMM also provides a full capability of relational algebra for the sophisticated users through a language called TAL (Table Algebra Language). A schematic diagram of the dialog/language used in the NVDBS is shown in the figure below. As shown below, the TAL is the internal language for the NDM. In fact, the internal structure of the NDM is a relational system, and the provision has been made to replace the RLQM by the TAL in a future "relational" NVDBS.

```
   TAB                                                |-------|
---------                                             |DMSREP |
| SCAN  |                                             |-------|
|       |                                                 ^
|       |-----|      RLQM        |-----|      RLQM        |
---->|  AUI |<------------->|AGENT|<-------------|
        |-----|     (TAL)        |-----|     (TAL)
           ^                        ^
           |                        |
       TAL |                        |
           |                        |TAL
           |        |-----|         |
           |---->|  NDM |<------|
                    |-----|
```

Figure 25. Schematic diagram of the dialog/language used in NVDBS

4.2 Design goals

The following three major goals govern the design and development of the DMM; the DMM should

* be easy to learn and simple to use,

- serve a wide portion of the user community,

- be extensible and adaptable for a continuously changing environment.

Research surrounding the Relational Model developed by E.F. Codd [CODD74] has shown that a data base more oriented toward the casual user is not only possible, but in many cases desirable. Results in [ZLOOF75] and [THOMAS75] indicated that the casual users could learn the "Query-by-Example" language in less than three hours and could then ask complicated queries as powerful as the first order predicate calculus. Our goal is therefore to design a relational system simpler than the "Query-by-Example," together with a training facility on which a casual user can learn to become a sophisticated, efficient user of the system in a few days.

With respect to the second principle, many have advocated that the architecture for the next generation DBMSs should provide both a non-procedural data language for the casual user and a procedural data language for the sophisticated user -- e.g. [NIJSSEN76]. One of the advantages of having these coexisting facilities is that the system has an easy to use language while still achieving higher efficiency through the procedural language facility. The DMM provides a simple man-machine dialog, TAB, for the casual user and the full power of relational algebra, TAL, for the sophisticated user.

The third goal has already been illustrated by the provision for TAL to take over the role of RLQM in a "future" relational NVDBS. The DMM provides a natural testbed for user acceptance of a future relational NVDBS. Once the relational approach is accepted by the user, DMM can be used as a full blown local DBMS at each node in a relational NVDBS.

4.3 Functional Components

The physical configuration of the AUI has been suggested in a previous section. The AUI has a core memory of 80k to 120k of 16 bit words, a half million to two million words of disk storage, a high speed swapping device, a non-impact printer with graphic capability, and a CRT terminal with cursor addressability and/or touch panel.

The DMM is implemented on the above physical configuration with the functional block diagram shown below. There are two major components: the TAB-to-TAL TRANSLATOR and the TAL INTERPRETER. The TRANSLATOR serves as a front-end process to the INTERPRETER and contains a machine dependent module for each different CRT terminal. This front-end process provides for easier integration of new CRT terminals introduced on the market. The INTERPRETER is the center of the DMM and has the full capabilities of relational algebra. The detailed specifications of the TRANSLATOR and the INTERPRETER are not included here. In the following sections, we will only discuss the distinct features of the simple man-machine interactive dialog (TAB) and the relational data language (TAL).

```
                      |----------|
|----|    TAB         |          |
|VCRT|<-------------->|TRANSLATOR|
|----|                |          |
                      |----------|
                          ^
                          |        |-----------|
                      TAL |        |           |    TAL     |-----|
                          |------->|INTERPRETER|<---------->| NDM |
                                   |           |            |-----|
                                   |-----------|
                                         ^
                                         |
                                         V
                                     |-------|
                                     | LOCAL |
                                     |STORAGE|
                                     |-------|
```

Figure 26. Functional Component of the DMM

4.4 A Tabular Man Machine Dialogue -- TAB

The TAB is designed for users who has no knowledge about the computer. It is our goal that such persons can use the system almost instantaneously and improve their skills through the assistance of the training facility provided by the system. To achieve this goal we utilize a CRT terminal and design a simple man-machine dialog presented in this section.

4.4.1 Virtual CRT Terminal

Before discussing TAB itself, let us first define the virtual CRT terminal (VCRT) where TAB is implemented. We assume that the VCRT has the following capabilities:

* Alphanumeric and graphic display,

- Zoning capability,
- Cursor addressability and/or touch panel, and
- Blinking and/or reverse video for field registration.

```
|-----------------------------------|----------|
|                                   | Command  |
|  Display Area                     | Area     |
|                                   |          |
|                                   |          |
|                                   |          |
|                                   |----------|
|                                   | X    INT | Signal
|                                   | RET  ?   | Area
|                                   |----------|
|                                   | Option   |
|                                   | Area     |
|                                   |          |
|-----------------------------------|----------|
```

Figure 27. Screen partitions of the VCRT

In general, the VCRT screen can be divided into four areas as shown in the above figure. The "display" area is an area for the user's keyboard interaction and for the system to display the results. The "command" and "option" areas are used for the system to display the commands and options which are available to the user at any point in time. To select a desired command or option, a user can touch that command or option area by finger (or light pen) and the system will then blink (or reverse video on) that command or option to indicate its acceptance. When a command is ready to execute, the system will display a "ready" message preceding the cursor to indicate its readyness. Otherwise, the system will always prompt for additional information. To execute a ready command, the user touches the "X" key in the "signal" area. Once the execution of a command is completed, the results will be displayed in the "display" area, and the commands and options for the next "TAB-transaction" will be displayed in their dedicated area.

Touching the "INT" in the "signal" area will interrupt the current operation of the TAB-transaction in progress and return to the final state of the previous "transaction". When the "RET" key in the "signal" area is touched, the system is returned to its parent state in the command hierarchy. The notion of the "command hierarchy" will become clear in the following discussion. The "?" in the "signal" area is used to ask any question or ask for any help from the system at any time.

4.4.2 Summary of the TAB Commands and Options

The basic information structure for the TAB is the "table". A table is a sequence of "rows", and each row is a sequence of values called "columns of that row" or column-values. Each "column" has a unique name with respect to the set of columns of a given table, and each table has a unique name with respect to the set of all tables. For example the following table (Fig. 28) contains three rows and each row has three columns. The name of the table is "Aircraft", and the names of the columns are "Airline/Aircraft#", "Aircraft-type", and "Capacity".

Aircraft

Airline/Aircraft#	Aircraft-type	Capacity
UA/130	B727	122
AA/431	B707	184
TWA/29	B747	291

Figure 28. table named 'aircraft'

There are three sets of commands in TAB: (a) single table commands, (b) multi-table commands, and (c) general macro commands. Three options are provided for the user to specify their desires on: (a) whether or not to eliminate duplications; (b) how many numbers of rows to be displayed, and (c) whether or not to save the TAL statements generated by the TRANSLATOR from the TAB original transaction. A summary of these commands and options is presented in figures 29 to 31.

COMMAND	PROMPT	ACTION
TABLE	table-name?	Display the contents of the specified table(s).
COLUMN	column-name?	Select the desired column(s).
ROW	aggregate-expression?	Select rows for which the aggregate expressions are satisfied.
EDIT	table-name?	Edit the table. If the table name did not exist, this will create a new table.
ASSIGN	name?	Assign a name to a table.
COPY	table_name? name?	Create a copy from an existing table, to be called "name".
REMOVE	table-name?	Remove table and "table-name".
SORT	order?	Sort the rows by the specified order.

Figure 29. Single-Table Commands

COMMAND	PROMPT	ACTION
JOIN	table-names? column_names?	Join two tables over a common column.
APPEND	table_names?	Append a table to another.
INTER-SECT	table_names?	Intersect two compatible tables.
SUB-TRACT	table_names?	Subtract all rows which appear on the other table.

Figure 30. Multiple Table Commands

OPTION	ALTERNATE	EXPLANATION
Duplication	YES, NO	To allow or not allow duplications.
Display	ALL, ANY FIRST(n) LAST(n) EVERY(n)	Display all, any, first n, last n, or every other n rows of the table.
SAVE	YES, NO	Save (or not) the internal TAL statements generated by the TRANSLATOR.

Figure 31. DMM Options

The best way to explain these commands and options is through examples. In order to explain the following examples, let us first define some notational convention:

@ denotes the cursor on the screen,

$ surrounding a word indicates the blinking of that word,

4.4.3 TAB Commands

Let us assume that a user logs into the AUI and the following information is displayed on the screen:

```
|------------------------------------|-----------|
| @                                  | LOGOFF    |
|                                    | NDM       |
|                                    | RLQM      |
|                                    | DMM       |
|                                    | VLM       |
|                                    |-----------|
|                                    | X    INT  |
|                                    | RET  ?    |
|                                    |-----------|
|                                    |           |
|                                    |           |
|                                    |           |
|------------------------------------|-----------|
```

Figure 32. AUI Login Mode

By touching the "DMM" in the commands area, the system will respond with a "ready" message preceding the cursor ("@") to indicate the readiness for execution of the DMM command. Now the user can signal the system to execute the DMM command by touching the "X" key in the "signal" area and the following results (Figure 33) will be displayed on the screen.

```
|-----------------------------------|-----------|
| Defaults:                         | SINGLE    |
|  Duplicate-- No                   | MULTIPLE  |
|  Display   -- First 2             | MACRO     |
|  Save      -- No                  |           |
| @                                 |           |
|                                   |-----------|
|                                   | X    INT  |
|                                   | RET  ?    |
|                                   |-----------|
|                                   | Duplicate |
|                                   | Display   |
|                                   | Save      |
|-----------------------------------|-----------|
```

Figure 33. DMM Mode

As shown on the screen, there are three commands and three options available for this "transaction." The defaults of the options are the following:

1. No duplication is allowed.

2. Only the first two rows of each table will be displayed on the screen.

3. Do not save the sequence of the TAL statements output by the TRANSLATOR.

The defaults may be modified at any time, e.g. the sequence of TAL statements may be saved, edited and then run during a later TAB-transaction, if so desired.

In the following discussions we assume that the entire example database is retrieved (under RLQM) and is stored locally with respect to the AUI.

4.4.3.1 <u>Single-Table Commands</u>

Let us now assume that in response to Figure 33, a user touches the "SINGLE" command and waits for a "READY" message in order to touch the "X" key. After "READY" from the system, and "X" from the user, the system will then enter the "SINGLE table" mode and the following information will be displayed on the screen (Figure 34).

```
|-----------------------------------|-----------|
| @                                 | TABLE     |
|                                   | COLUMN    |
|                                   | ROW       |
|                                   | EDIT      |
|                                   | ASSIGN    |
|                                   | COPY      |
|                                   | REMOVE    |
|                                   | SORT      |
|                                   |-----------|
|                                   | X    INT  |
|                                   | RET  ?    |
|                                   |-----------|
|                                   | Dup - NO  |
|                                   | Dis - F 2 |
|                                   | Save- NO  |
|-----------------------------------|-----------|
```

Figure 34. Single Table Mode

Seven commands and three options (with default values specified) are available in the single table mode. The detailed descriptions of these commands follow in the paragraphs below.

4.4.3.1.1 <u>TABLE</u> <u><table name list></u>

The TABLE command displays the contents of the specified tables. For example, if a user touches the TABLE command in response to the Figure 34, then the system will prompt for "table-name?" as shown in Figure 35. (Note that the TABLE command is blinking to indicate the acceptance of the command).

The user may answer this request by typing an "all" (or "*") for all tables in the data base to be displayed. In response to the user, a "ready" message will be displayed on the screen to indicate that the system is ready to execute the command. At this time, the user may touch the "X" key to initiate the execution of the TABLE command.

```
|-----------------------------------|-----------|
| table_name? @                     |$TABLE$$$$$|
|                                   | COLUMN    |
|                                   | ROW       |
|                                   | EDIT      |
|                                   | ASSIGN    |
|                                   | COPY      |
|                                   | REMOVE    |
|                                   | SORT      |
|                                   |-----------|
|                                   | X    INT  |
|                                   | RET  ?    |
|                                   |-----------|
|                                   | Dup - NO  |
|                                   | Dis - F 2 |
|                                   | Save- NO  |
|-----------------------------------|-----------|
```

Figure 35. TABLE ready

The following is the result of the above TABLE command (figure 36):

```
--------------------------------------------------------------------
| table-name? all
| ready.
|  Aircraft
|   |------------------|--------------|---------|
|   |Airline/Aircraft# |Aircraft-type |Capacity |
|   |------------------|--------------|---------|
|   |     UA/130       |    B727      |  122    |
|   |     AA/431       |    B707      |  184    |
|                                      1 row left.
|  Airport
|   |---------|------------------|---------------|
|   |Mnemonic |  Airport-Name    |     City      |
|   |---------|------------------|---------------|
|   |  SFO    |S.F.International |San Francisco  |
|   |  LAX    |L.A.International |Los Angeles    |
|                                      3 rows left.
|  Personnel
|   |----------------|----------|-----------------|
|   |   Name         |Employee# |Years_Experience |
|   |----------------|----------|-----------------|
|   |Elise Granville | 11239    | 7               |
|   |Scarlet O'Hara  | 67349    | 9               |
|                                        3 rows left.
|  Specialty
|   |---------------------|
|   |Type|  Title         |
|   |---------------------|
|   |Tech|Captain         |
|   |Tech|First_Officier  |
|             3 rows left.
|  Flight
|   |---------------|----------|------------|-------|------------|
|   |Airline/Flight#|#1st_Pass |#Coach_Pass |Origin |Destination |
|   |---------------|----------|------------|-------|------------|
|   |     UA/86     | 14       | 122        | SFO   | BOS        |
|   |     TWA/141   | 22       | 230        | LAX   | PHX        |
|                                                    2 rows left.
| @
|
|
```

Figure 36. TABLE all

Note that only the first two rows of each table are displayed on the screen as specified by the display option. However, the number of rows in each table which are not displayed is specified at the right-lower corner of each table. Note also that if the results cannot fit into one screen, the remaining portions can be displayed by "rolling" up/down or left/right on the screen.

4.4.3.1.2 COLUMN <column name list>

The COLUMN command forms a new table by selecting the desired columns out of a given table, as specified by the user. For example say the user wants to select the "Airline/Aircraft#" and the "Aircraft_name" from the table "Aircraft". The user can do so by first touching the COLUMN command and then touching the two desired columns in response to the "column_name?" request. In response to each of these actions, the system blinks the COLUMN command and the selected column names to indicate acceptance of those commands. At the same time, the system also displays the "ready" message on the screen. If the user wants additional columns, he can touch those columns, otherwise he can touch the "X" key to initiate the execution of the COLUMN command. Here, we assume the user touches the "X" signal immediately, and the result appears in figure 37 below.

```
|--------------------------------------------------------------
| column_name?
| ready.
|  Temp_1
|   |------------------|--------------|
|   |Airline/Aircraft# |Aircraft_type |
|   |------------------|--------------|
|   |     UA/130       | B727         |
|   |     AA/431       | B707         |
|                            1 row left.
|  @
|
```

Figure 37. COLUMN Command

4.4.3.1.3 ROW <aggregate expression list>

The ROW command forms a table by selecting those rows (of the given table) which satisfy the specified aggregate expressions. The detailed explanation of the aggregate expression is presented in paragraph 4.4.5.

In the following example, let us assume that the Flight table (figure 38) is the current table on the screen.

```
|-----------------------------------------------------------
| Table_name? Flight
| ready.
|  Flight
|   |---------------|---------|-----------|------|-----------|
|   |Airline/Flight#|#1st_Pass|#Coach_Pass|Origin|Destination|
|   |---------------|---------|-----------|------|-----------|
|   |     UA/86     | 14      | 122       | SFO  | BOS       |
|   |    TWA/141    | 22      | 230       | LAX  | PHX       |
|                                                2 rows left.
| @
```

Figure 38. TABLE Flight

All the flights originating from "SFO" can be selected by using the ROW command with the aggregate expression as shown in the following figure.

```
|-----------------------------------------------------------
| Aggregate_expression?
|  Flight
|   |---------------|---------|-----------|------|-----------|
|   |Airline/Flight#|#1st_Pass|#Coach_Pass|Origin|Destination|
|   |---------------|---------|-----------|------|-----------|
|   |               |         |           | =SFO |           |
| ready. @
|
```

Figure 39. Flight Origin is equal to SFO

The above aggregate expression specifies that the value of the "origin" must be equal to "SFO". The result of the above query is:

```
|------------------------------------------------------------------
| Temp_2
|  |----------------|---------|-----------|------|-----------|
|  |Airline/Flight# |#1st_Pass|#Coach_Pass|Origin|Destination|
|  |----------------|---------|-----------|------|-----------|
|  |     UA/86      | 14      | 122       | SFO  | BOS       |
|  |     AA/833     | 38      | 382       | SFO  | IAD       |
|  |----------------|---------|-----------|------|-----------|
|                                                   0 row left.
| @
|
```

Figure 40. Origin row equal to SFO (*)

If the user only wants to know the flight number of those flights originating in "SFO", then the user can touch the "Airline/Flight#" column before or after typing the aggregate expression as shown in the following figure.

```
|------------------------------------------------------------------
| Aggregate_expression?
|  Flight
|   |----------------|---------|-----------|------|-----------|
|   |Airline/Flight# |#1st_Pass|#Coach_Pass|Origin|Destination|
|   |----------------|---------|-----------|------|-----------|
|   |$$$$$$$$$$$$$$$$|         |           | =SFO |           |
| ready. @
|
```

Figure 41. Airline/Flight# originating from SFO: query

The result of this query is displayed in figure 42.

(*) Note that the table Temp_2 (figure 40) has only two rows, therefore the table bottom line is displayed to indicate that the table is complete.

```
|------------------------------------------------------------
|
|  Temp_3
|    |---------------|
|    |Airline/Flight#|
|    |---------------|
|    |     UA/86     |
|    |     AA/833    |
|    |---------------| 0 row left.
| @
|
```

Figure 42. Airline/Flight# originating from SFO: result

4.4.3.1.4 <u>EDIT</u> <*table name*>

If table name is a new name, not yet associated with any table, then EDIT will create a new table and prompt the user to obtain its structures and constraints. EDIT is a complete text and table editor which contains all features of a text editor plus features for table edition. The commands and options available under the EDIT command are shown in figure 43.

```
|-------------------------------------|-----------|
| Defaults--                          | SEARCH    |
|  Mode       -- COLUMN               | INSERT    |
|  Occurrence-- FIRST                 | DELETE    |
|  Direction -- FORWARD               | CHANGE    |
| @                                   | REPEAT    |
|                                     |           |
|                                     |-----------|
|                                     | X    INT  |
|                                     | RET  ?    |
|                                     |-----------|
|                                     | Mode      |
|                                     | Occurrence|
|                                     | Direction |
|-------------------------------------|-----------|
```

Figure 43. Screen for the Edit Command

As shown on the screen, there are five commands and three options available under the EDIT command. The defaults of these options are: column mode, first occurrence only, and forward direction. The explanations of these commands and options are presented in the following two tables.

COMMAND	PROMPT	ACTION
SEARCH	pattern?	Search for rows which match the given pattern.
INSERT	values?	Insert a row.
DELETE		Delete a row.
CHANGE	column? values?	Change the column value to the given value.
REPEAT	times?	Repeat a sequence of commands a number of times.

Figure 44. EDIT Commands

OPTION	ALTERNATE	EXPLANATION
Mode	COLUMN, TABLE	Column or table mode.
Occurrence	FIRST, ALL	Effect the first (or all) occurrence(s).
Direction	FORWARD, BACKWARD	Effect the forward (or backward) direction.

Figure 45. EDIT Options

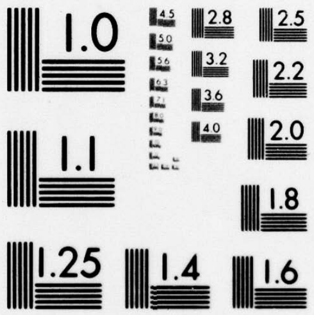
Similar to [INGRES76/77], there are five special characters which are powerful tools in the specification of a search pattern. The meaning of these five characters is defined as follows:

*	matches any string of characters.
?	matches any single character.
[...]	matches any single character belonging to the set in the brackets.

AD-A074 205 FORD AEROSPACE AND COMMUNICATIONS CORP PALO ALTO CA F/G 9/2
PRELIMINARY DESIGN OF A NETWORK VIRTUAL DATA BASE SYSTEM.(U)
AUG 79 F30602-77-C-0158
UNCLASSIFIED RADC-TR-79-102 NL

2 OF 2
AD
A074205

END
DATE
FILMED
10-79
DDC


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

" when used before one of these five characters (including "
itself) will disable the above described mechanism.

For example, the pattern ?[a-k]* matches any string whose second character is either A, B, C, ..., or K.

The SEARCH, INSERT, DELETE, and CHANGE are defined in the obvious manner. The SEARCH command will eventually be extended to select column values through predicate expressions using the aggregate expressions defined in Section 4.4.5. The system provides a facility which allows the user to accumulate a sequence of commands and then execute them once through the touching of the "X" signal. If the "REPEAT n times" is the last command of the defined sequence of commands, then the execution of that sequence of commands will be repeated n times.

The "Mode" option is used to specify the operational domains of the commands. The "Column mode" restricts the commands to the specified columns only, and the "Table mode" applies to the entire table. The "Occurence" option specifies whether the commands apply to the first occurence of the matching pattern or to all occurences of the matching patterns. The "Direction" option applies to the SEARCH and the INSERT commands only.

4.4.3.1.5 ASSIGN <name> to <table name>

The ASSIGN command assigns a name to an existing table called table_name. If the name happens to be a name for a pre-existing table, a warning message will be issused before the table is destroyed. In any case, the name becomes an alias of the pre-existing table called table_name. Note that the modification to the table using the alias name will change the table called table_name. A COPY command is provided for creating a copy of the pre-existing table.

4.4.3.1.6 COPY <name> to <table name>

This command is used to create a new copy of an existing table called table-name and to assign the name to the newly created copy.

4.4.3.1.7 REMOVE <table name>

This command is used to remove an existing table and its name.

4.4.3.1.8 SORT <table name> on <column order list>

This command will sort the rows of the given table according to the <column_order_list>. The <column_order_list> is a list of the column_order which is a concatenation of column_name and the {ASC, DES} values. Each column_order specifies that the values of that column are to be sorted in either ascending or descending order. The first column_order in the list is the major sort key and the remaining column_orders are the minor keys in the order of their appearances on the list.

4.4.3.2 Multiple Table commands

If in response to figure 33, the user wants to use the MULTIPLE-table command, the user touches the MULTIPLE-table command and then the "X" signal to get the following response on the screen.

```
|---------------------------------|-----------|
| @                               | JOIN      |
|                                 | APPEND    |
|                                 | INTERSECT |
|                                 | SUBTRACT  |
|                                 |           |
|                                 |-----------|
|                                 | X     INT |
|                                 | RET   ?   |
|                                 |-----------|
|                                 | Dup - NO  |
|                                 | Dis - F 2 |
|                                 | Save- NO  |
|---------------------------------|-----------|
```

Figure 46. MUTIPLE tables submode

Note that the values of the options were specified during the previous transaction and can be changed at any time, if it is so desired.

4.4.3.2.1 JOIN <table names> over <column names>

The JOIN command can "join" two tables which possess what we call loosely a "common column", to form a new table. The result of the JOIN command is a table which consists of a concatenation of a row from one table and a row from the other table where these two row have the same values in their "common column". For example, if one wants to join the tables "Flight_Origin" and "Airport" over the "Origin" column of the "Flight_Origin" table and the "Mnemonic" column of the "Airport" table, we can touch the JOIN command and then answer the prompt "table_names?" with "Flight_Origin" and "Airport". The system will display the structure of these tables and then ask for the "common column" to be joined over as shown in the following figure.

```
 --------------------------------------------------------------
| table_names? Flight_Origin, Airport
|
|  Flight_Origin
|   |---------------|------|
|   |Airline/Flight#|Origin|
|   |---------------|------|
|   |               |      | 2 rows left.
|
|  Airport
|   |--------|----------------|-------------|
|   |Mnemonic|  Airport-Name  |    City     |
|   |--------|----------------|-------------|
|   |        |                |             | 3 rows left.
| column_names? @
```

Figure 47. JOIN two tables

In response to the above "column_names?" prompt, the user can touch the "Origin" column of the "Flight_Origin" table and the "Mnemonic" column of the "Airport" table. The system is then ready to execute the JOIN command, and a "ready" message is displayed on the screen. If there is nothing to be changed, the user can touch the "X" signal in the signal area to initiate the execution of the JOIN command. The following is the result of these operations.

```
|--------------------------------------------------------------
|
|  Temp_4
|   |---------------|------|----------------|-------------|
|   |Airline/Flight#|Origin|  Airport_name  |    City     |
|   |---------------|------|----------------|-------------|
|   |     UA/86     | SFO  |S.F.International|San Francisco|
|   |    TWA/141    | LAX  |L.A.International|Los Angeles  |
|                                                2 rows left.
| @
```

Figure 48. JOIN Flight_Origin and Airport

The above example is the join on equal values and sometimes called "equal_JOIN". For unequal_JOINs the user types the following expression in response to the "column_names?" prompt:

```
   <table1_name>.<column1_name> op <table2_name>.<column2_name>
     where "op" stands for any relational operator among the following,
             >   greater than,
             <   less than,
             >=  greater than and equal to,
             <=  less than and equal to,
             =   equal to,
             ~=  not equal to.
```

4.4.3.2.2 <u>APPEND</u> <u><table name></u> <u>to</u> <u><table name></u>

The APPEND command adds one table to the end of another table provided that the tables have compatible structures. The result is a table containing all rows of both tables. Note that inserting a large number of new rows into a table can be achieved by either the APPEND command or by the INSERT command under the EDIT command described in paragraph 4.4.3.1.4.

For example, let us append the table "New_Aircraft" to the table "Aircraft" as shown in the Figure 49.

Aircraft

Airline/Aircraft#	Aircraft-type	Capacity
UA/130	B727	122
AA/431	B707	184
TWA/29	B747	291

New_Aircraft

Airline/Aircraft#	Aircraft-type	Capacity
PAM/123	B727	122
JAL/101	DC10	210

Figure 49. Tables Aircraft and New_Aircraft

The user can touch the APPEND command and then answer the prompts appropriately as shown in the following figure.

```
|-----------------------------------|----------|
| table_names? New_Aircraft, Aircraft| JOIN     |
|  APPEND New_Aircraft to Aircraft. |$APPEND$$$$|
| ready.@                           | INTERSECT |
|                                   | SUBTRACT |
|                                   |          |
|                                   |----------|
|                                   | X    INT |
|                                   | RET  ?   |
|                                   |----------|
|                                   | Dup - NO |
|                                   | Dis - F 2|
|                                   | Save- NO |
|-----------------------------------|----------|
```

Figure 50. APPEND command

Note that the "option" area indicates that no duplication is allowed and only displays the first two rows of each table. Assume that the user wants to have duplications and to display all rows in order to check the result of the APPEND command. The user can change these two options by touching them and answer the prompts as shown in the following figure.

```
|-----------------------------------|----------|
| table_names? New_Aircraft, Aircraft| JOIN     |
|  APPEND New_Aircraft to Aircraft. |$APPEND$$$$|
| Duplication? Yes                  | INTERSECT |
| Display? all                      | SUBTRACT |
| ready. @                          |          |
|                                   |----------|
|                                   | X    INT |
|                                   | RET  ?   |
|                                   |----------|
|                                   |$Dup - $$$$|
|                                   |$Dis - $$$$|
|                                   | Save- NO |
|-----------------------------------|----------|
```

Figure 51. Change options

After specifying the changes to the options, the user now can execute the APPEND command by signaling "X" on the screen. The result of the above commands is illustrated in the following figure:

```
|------------------------------------------------------------
|
|  Temp_5
|    |----------------|-------------|--------|
|    |Airline/Aircraft#|Aircraft-type|Capacity|
|    |----------------|-------------|--------|
|    |     UA/130     |    B727     |  122   |
|    |     AA/431     |    B707     |  184   |
|    |     TWA/29     |    B747     |  291   |
|    |     PAM/123    |    B727     |  122   |
|    |     JAL/101    |    DC10     |  210   |
|    |----------------|-------------|--------|
|                                    0 row left.
| @
|
```

Figure 52. APPEND New_Aircraft to Aircraft

4.4.3.2.3 INTERSECT <table names>

The intersection of two compatible tables (i.e., the tables which have compatible structures) results in a table containing the common rows of both. *For example, to intersect the following two tables:*

Aircraft

Airline/Aircraft#	Aircraft-type	Capacity
UA/130	B727	122
AA/431	B707	184
TWA/29	B747	291

Old_Aircraft

Airline/Aircraft#	Aircraft-type	Capacity
PAM/123	B727	122
UA/130	B727	122

Figure 53. Tables Aircraft and Old_Aircraft

The user can touch the INTERSECT command and then type these two table names in response to the question of "table_names?".

```
|-----------------------------------|-----------|
| table_names? Old_Aircraft, Aircraft| JOIN      |
|  INTERSECT Old_Aircraft, Aircraft. | APPEND    |
| ready.@                           |$INTERSECT$|
|                                   | SUBTRACT  |
|                                   |           |
|                                   |-----------|
|                                   | X    INT  |
|                                   | RET  ?    |
|                                   |-----------|
|                                   | Dup - YES |
|                                   | Dis - ALL |
|                                   | Save- NO  |
|-----------------------------------|-----------|
```

Figure 54. INTERSECT command

After the system is ready to execute and "X" is signaled, the following screen will be shown:

```
|------------------------------------------------------------
|
| Temp_6
|   |-----------------|-------------|--------|
|   |Airline/Aircraft#|Aircraft-type|Capacity|
|   |-----------------|-------------|--------|
|   |     UA/130      |    B727     |  122   |
|   |-----------------|-------------|--------|
|                                       0 row left.
| @
|
```

Figure 55. INTERSECT Aircraft and Old_Aircraft

4.4.3.2.4 SUBTRACT <table1 name> from <table2 name>

The SUBTRACT command is used to delete rows of a given table which appear in the other table provided that both tables have compatible structures. Thus, for example, to delete the "Old_Aircraft" from the "Aircraft" table (as shown in Figure 53), one can touch the SUBTRACT command and then respond to the "table_names?" prompt with these two table names. Note that the system always displays a message to clearly indicate which table will be used to delete the other table. This allows the user to double check his request before the actual deletion takes place.

```
|-------------------------------------|-----------|
| table_names? Old_Aircraft, Aircraft | JOIN      |
|  SUBTRACT Old_Aircraft from Aircraft.| APPEND   |
| ready.@                             | INTERSECT |
|                                     |$SUBTRACT$$|
|                                     |           |
|                                     |-----------|
|                                     | X    INT  |
|                                     | RET  ?    |
|                                     |-----------|
|                                     | Dup - YES |
|                                     | Dis - ALL |
|                                     | Save- NO  |
|-------------------------------------|-----------|
```

Figure 56. SUBTRACT command

If this is the user's intention, the user can touch the "X" key to trigger the execution of the SUBTRACT command. The result of the above deletion is shown in the following figure.

```
|-------------------------------------------------------------
|
|    |-----------------|-------------|--------|
|    |Airline/Aircraft#|Aircraft-type|Capacity|
|    |-----------------|-------------|--------|
|    |     AA/431      |    B707     |  184   |
|    |     TWA/29      |    B747     |  291   |
|    |-----------------|-------------|--------|
|                                     0 row left.
| @
|
```

Figure 57. SUBTRACT Old_Aircraft from Aircraft

Similar to the insertion, rows can be deleted by the SUBTRACT command or by the DELETE command under the EDIT command (defined in Section 3.4.1.4).

4.5 MACRO FACILITY

The macro facility is provided as a way for sophisticated users to define their own commands and to effectively use the full power of the relational algebra. Almost every operating system today has a general purpose macro facility, therefore the DMM will adapt these facilities for its MACRO command. The specifications of any current operating system's macro facility satisfies our needs here and therefore the facility will only be discussed briefly.

There are two commands available in the MACRO mode as shown in Figure 58.

```
|------------------------------------|-----------|
| @                                  | DEFINE    |
|                                    | EXECUTE   |
|                                    |           |
|                                    |           |
|                                    |-----------|
|                                    | X    INT  |
|                                    | RET  ?    |
|                                    |-----------|
|                                    | Dup - YES |
|                                    | Dis - ALL |
|                                    | Save- NO  |
|------------------------------------|-----------|
```

Figure 58. MACRO mode

The DEFINE command invokes a standard general purpose macro facility to define macros of TAL statements. The EXECUTE command invokes the same macro facility to expand the given macro definitions and then pass them to the INTERPRETER for direct execution. The EXECUTE command can also be used to execute a dialog file saved by the SAVE option.

4.6 AGGREGATE EXPRESSION

Aggregate expressions provide ways to group, accumulate, or total over a set of values. An aggregate expression is a combination of arithematic expressions and aggregate functions listed below:

- COUNT — Count of occurrence.
- SUM -- Summation.
- AVG — Average.
- MAX — Maximum.
- MIN -- Minimum.
- ANY -- Any row.
- ALL -- All rows.
- FIRST (n) -- First n rows.
- LAST (n) -- Last n rows.
- EVERY (n) -- Every other n rows.

For example, the following aggregate expressions

table_x

A	B	C	D	E
x	>x	y FIRST(5)	z EVERY(2)	< SUM(x,y,z)

Figure 59. Aggregate Expressions

define that:

a. the variable x denotes all values of column A;

b. the qualification ">x" specifies that the value of column B must be greater than x (i.e., the value of the column A of the same row);

c. the variable y denotes the first five rows of column C;

d. the variable z denotes every other values of column D;

e. the qualification "< sum(x,y,z)" specifies that the values of column E must be less than the summation of the values of x, y, and z of the same row.

4.7 A TABLE ALGEBRA LANGUAGE -- TAL

4.7.1 Table

A "table" is like a "normalized" relation in a relational system except that:

a. It may have duplications (.i.e., identical rows).

b. The rows are ordered.

Two operators, NODUP and SORT, are provided for the above two execeptions. The NODUP operator is used to suppress the duplications from a table, and the SORT operator sorts the rows in a table according to a given order. A table with all duplications suppressed is equivalent to a normalized relation in the sense that every column value in a table is a datum. In other words, there is no table of tables.

A table has a unique name and each column within a table must have a unique name, too. The concatenation of table_name and column_name constitutes the full name of a column, e.g.

<table_name>.<column_name> --> <full_column_name>

For computational convenience, a table may be specified dynamically by a list of rows of constants. An example of such a list is shown in the following:

<3,5,7.1,"on">, <2,5,6.3,10,"off">, <9,5,0.0,2,"no">

It is called a "constant table". Similar to the constants in the programming language, the constant tables do not have names, and can only be specified explicitly.

4.7.2 Table Operator

The "table_operator" is an operator which takes one or two tables as its operand(s) and produces a table as its result. Corresponding to the SINGLE table commands in TAB, we have the following five unary table_operators. Each unary table_operator takes one table (the default value is the current table) together with the optional parameter list as defined in the following table.

OPERATOR	OPERAND	PARAMETER_LIST
TABLE	<table>	ALL, ANY, FIRST (n), LAST (n), EVERY (n)
COLUMN	<table>	<column_name_list>
ROW	<table>	<aggregate_expression_list>
ASSIGN	<table>	<name>
COPY	<table>	<name>
SORT	<table>	<order_list>
REMOVE	<table>	None
NODUP	<table>	None

Figure 60. Unary Table_Operators

Similary, there are four binary table_operators (fig. 61) corresponding to the four commands available under the MULTIPLE table mode of the TAB. Each binary table_operator takes two tables as its operands and has an optional parameter list as specified in the following table.

OPERATOR	OPERANDS	PARAMETER_LIST
JOIN	<table1>,<table2>	<column_name_list>
APPEND	<table1>,<table2>	None
INTERSECT	<table1>,<table2>	None
SUBTRACT	<table1>,<table2>	None

Figure 61. Binary Table_Operators

4.7.3 Table Expression

The "table_expression" is a string of table_names and table_operators which satisfies the production rules examplified below. The table_expression evaluates to a table. The major syntactical definitions of the table_expression are summarized in the following BNF production rules.

```
<table_expression> ::= <table> <unary_op> <parameter_list>
                     | <table> <binary_op> <table> <parameter_list>

<table> ::= <table_name>  |  <constant_table>
          |  ( <table_expression> )

<unary_op> ::= TABLE  |  COLUMN  |  ROW  |  ASSIGN
             |  COPY  |  SORT  |  REMOVE  |  NODUP

<binary_op> ::= JOIN  |  APPEND  |  INTERSECT  |  SUBTRACT
```

Note that the result of a table_expression is a table, so that the result can be an operand of another table_operator. The following are examples of table_expressions:

1. (Flight ROW Origin=SFO) COLUMN Airline/Flight#

The result of this expression is a one column table of Airline/Flight#(s) of all flights originating from SFO as shown in Figure 42. Note that the subexpression (Flight ROW Origin=SFO) selects all rows from the "Flight" table whose "Origin" column contains the value SFO. The result of this subexpression is shown in Figure 40. Note that a temporary name, Temp_2, is assigned to this table by the system (as shown in the figure). The COLUMN operator in the original expression takes the temporary table, Temp_2, as its operand and suppresses all columns except the Airline/Flight# column (as shown in Figure 42). The same example is presented in another form (see Section 4.4.3.1.3) by using the TAB language.

2.

```
((( Flight_Origin ASSIGN
   ( Flight COLUMN Airline/Flight#,Origin) )
   JOIN Airport on Flight_Origin.Origin = Airport.Mnemonic )
            ROW City="San Francisco" )
                  COLUMN Airline/Flight#
```

The result of this rather complex expression is a one column table of Airline/Flight#(s) of all flights from San Francisco. In order to explain this expression, let us decompose this expression into the following four subexpressions and then evaluate them step by step.

a.

```
( Flight_Origin ASSIGN
     ( Flight COLUMN Airline/Flight#,Origin) )
```

This subexpression forms a new table called "Flight_Origin", by selecting the "Airline/Flight#" and "Origin" columns from the "Flight" table. The structure of the "Flight_Origin" table is shown in Figure 47.

b.

```
( Flight_Origin JOIN Airport
     on Flight_Origin.Origin = Airport.Mnemonic )
```

This subexpression joins the tables "Flight_Origin" (i.e., the resulting table of the previous subexpression) and "Airport" on those rows which have the same values in the "Origin" column of the "Flight" table and the "Mnemonic" column of the "Airport" table. The result of this subexpression is shown in Figure 48. Note that "Temp_4" is the system assigned name for this table, and we will also use this name for the following discussions.

c. (Temp_4 ROW City="San Francisco")

This subexpression forms a table by selecting those rows in the "Temp_4" table whose column value of the "City" column is "San Francisco". The following figure is the result of this subexpression.

```
Temp_7
 |---------------|------|----------------|-------------|
 |Airline/Flight#|Origin|  Airport_name  |    City     |
 |---------------|------|----------------|-------------|
 |     UA/86     | SFO  |S.F.International|San Francisco|
 |     AA/833    | SFO  |S.F.International|San Francisco|
 |---------------|------|----------------|-------------|
                                               0 row left.
```

Figure 62. Temp_4 ROW City=San Francisco

For convenience of our discussion, let us call this table "Temp_7".

d. Temp_7 COLUMN Airline/Flight#

This subexpression suppresses all columns except the "Airline/Flight#" column from the "Temp_7" table as shown in Figure 62. The result of this subexpression (as shown in the Figure 42) is the result of the original expression.

4.8 CONCLUSIONS AND DISCUSSIONS

The Main objective of the DMM is to provide a simple but efficient facility for user to manipulate data stored in the local storage. This goal is achieved by providing the user with a simple man-machine dialog in pictorial form (the TAB language) and a powerful relational algebra (the TAL language). The simplicity of the TAB has been demonstrated through the examples in this section. TAB is very elegant as it has the full power of relational algebra but with a relatively small set of commands and options. The data under the TAB is also easy to comprehend because all data are displayed in table form. The ability to display all permissible commands and options on a screen and to prompt all necessary parameters directly, greatly minimizes the necessary learning and understanding of the idiosyncrasies of each command and option. If the SAVE option is used, all TAL statements generated from a sequence of TAB commands will be saved into a command file. When a batch of commands is presented (in a file) to the DMM, an optimizer will be used to minimize the number of operations and the number of access to the data base. The user can always take this advantage to use the DMM efficiently. Another advantage of the DMM is the natural progression of the languages from TAB to TAL, where the TAB user can learn and master the TAL in a few days.

The following two areas are planned for the future extension of the DMM. The first area is to add a Data Definition Language (DDL) to the DMM. Currently, a DDL is not needed for the DMM because all data were defined and later extracted from the NVDB by the RLQM. However, provisions have been made for adding a DDL into the DMM to form a self-contained relational DBMS. This relational DBMS will be used to replace the RQLM in a future relational NVDBS. The second area of the extension is to enhance the EDIT command of the TAB language, so that a more sophisticated query (equivalent - in power - to the first order predicate calculus) can be used in conjunction with the SEARCH and CHANGE subcommands.

5. CONCURRENCY CONTROL

This section discusses an approach to satisfy the concurrency control requirements of the NVDBS.

Note that only the NVDBS transactions are considered here, and the possible deadlocks within a given DBMS, hidden from the AGENTs by its DMSREP, are beyond the scope of this section.

This section contains, first, some general notions about a decentralized detection scheme based on digraphs. Then, a decentralized detection scheme based on eventcounts is detailed -- see for instance [GRAPA76] [LAMPORT76] [LELANN78] [REED76] [ELLIS77] [BADAL78]. A decentralized solution clearly leads to a less vulnerable but less efficient protocol. Therefore, it may later be necessary to study a trade-off [ALSBERG].

Aside from the decentralized versus centralized trade-off, our approach deals with the radical choice between preventive and cure treatment for the design of a concurrency control mechanism (*). Such a choice is discussed now. In general, there are two manners to handle potential deadlocks in the allocation of resources in a system. One is to detect these deadlocks after they have taken place, and the other is to prevent any transaction from being initiated if there is any possibility for a deadlock to occur. In the context of centralized system, the view has long been held that since rollback and recovery were needed anyway for handling malfunctions of all sorts of origin, the cure approach was almost free. Indeed, only a detection mechanism is needed, and it is needed in both approaches: prevention and cure. Furthermore, many operational commercial DBMSs show statistics with very low

(*) "Mieux vaut prevenir que guerir" old French proverb.

frequency of deadlock occurrences [GRAY78].

Such an argument was acceptable in the context of centralized system. However, there are several reasons why the prevention approach may be judged a better one in our context of a distributed system. First, consider NVDBS principle #1 of minimal interference with the member DBMS (which is branched on the NVDBS). If the NVDBS concurrency control was the curative sort, any kind of rollback provoked by an NVDBS AGENT would have to be followed in a consistent manner by the concerned DMSREPs. This is to say that the DMSREPs should have the power to decide a rollback of their respective DBMSs. This is not the ordinary user's capability for the DMSREP, and goes against the NVDBS principle #1, which would like to have the DMSREP appear as an ordinary user to its corresponding DBMS.

A second reason (for preferring prevention to cure) is that different rollbacks from different DMSREPs may be in conflict unless the most stringent synchronization exists between the different modules of the NVDBS. Such a synchronization may be expensive to achieve. Yet it would be necessary to propagate rollbacks to the virtual or network level of the NVDBS.

On the other hand, avoidance of deadlocks or prevention is a consistent solution which may be applied everywhere, with a minimum of surprises. The only problem with preventive care is that data dependent access is very difficult to deal with. Of course, the record level query mode is relatively limited in terms of data dependent access, which makes our position easier at this early stage of the NVDBS design. But later stage powerful relational queries would certainly include data dependent access. Actually, one may well argue that data dependent access is the essential manner in which decentralization will be accomplished in the NVDBS. A transaction which includes data dependent access will be treated by the concurrency controller as accessing all possible sites (possible from the viewpoint of that data dependent query, not necessarily all sites), an expensive protective attitude which may clog the system if it occurs frequently.

The truth is that if the NVDBS is to allow for all sorts of DBMSs to branch on its network, there is bound to be some DBMS which rollbacks now and then, with some possible ill effect on the integrity and consistency of the overall virtual database (NVDB). Worse, two DBMSs may demand different roll-backs simultaneously, thereby necessitating synchronization at the NVDBS, and the power for the NVDBS to rollback all DBMSs through their DMSREPs. But some DBMSs may not have this rollback capability available to their DMSREPs: and in view of the NVDBS principles, having the NVDBS provide such capability is out of the question. This suggests that some DBMS may be more or less disturbed as members of the NVDBS, depending upon the greater or lesser power of their particular user, the DMSREP. Some DMSREP may not allow any update request from the AGENT because of lack of recovery facility of its local DBMS or because of the lack of concurrency control of the DBMS, e.g. the DMSREP cannot lock the DBMS within the framework of the particular DBMS (recall principle demanding that REP look like ordinary user); thus the NVDBS transaction which would contemplate updating a node without safe interface may be rejected by the DMSREPs concerned.

This section will attempt to present an approach which is both flexible and resilient enough to constitute a profitable basis for further work. It borrows heavily on previous work in the field [ELLIS77]. In the next para-graph, we shall discuss the concept of control subdigraphs. It provides for a very general solution, especially valid in the case of a geographically distributed network. Then some protocols are proposed, utilizing eventcounts or timecounts in order to linearize certain events subsequences, thereby guaranteeing a deadlock-free operation.

Again, our approach is preventive, and therefore penalizes data-dependent access in favor of transactions involving strictly data-independent accesses.

5.1 DECENTRALIZED DEADLOCK DETECTION THROUGH DIGRAPHS

5.1.1 Definitions This is an introduction to the raw problem of concurrency control in a distributed (non centralized) data management system such as the NVDBS. In this section we present a deadlock detection scheme to be used between the different modules (AGENTs, REPs, NDMs) of the NVDBS. The solution proposed in this section is one which does not require any "timestamp" mechanism.

Conceptually, we define a control digraph as a digraph where nodes represent transactions and an oriented edge from node t1 to node t2 means that transaction t1 is blocked by its own request for resource held by transaction t2. A few definitions are in order; they will also be useful for the next section:

1. A "transaction" is a portion of code delimited by BEGIN_TRANS and END_TRANS. A transaction may originate or start executing at one network component (site) but may not remain local to this component. It may be a few instructions, or an entire process. Conceptually, it isolates a group of actions between two consecutive "consistent states" of the whole virtual database. As noted above, the different accesses necessitated by

the transaction are known in advance. Data dependent accesses are treated as if they necessitated all possible accesses under the circumstances, before execution time. Within a transaction, the following rule holds. Once an access authorization is given and the corresponding lock is set, no release is executed until the end of the transaction. Exception: explicit release command. Such a command is available to the system or the sophisticated user when they know that the consistency of the database is not affected. A particularly effective use of such a command may be made after a "data dependent access" is completed at execution time, so as to avoid the monopolization of large amount of resources for too long a period of time.

2. A "resource" is typically any access to a portion of the database, or an event or a particular message to emanate from some process. A transaction may require a resource and immediately obtain it, or it may be blocked until the completion of another transaction which holds the resource. We assume that the NVDBS modules have a list of all resources and their availability, at the module level. Each NVDBS module has control of a set of resources. Typically, a transaction requires input resources which are included in the transaction "domain". This domain is the minimal set of lockable resources which contain the required resources, and it is larger or smaller depending upon the less or more refined granularity of the locking mechanism. A transaction also requires a set of output resources which are gathered into what is called the "range" of the transaction. Again, the range is the minimal set of resources which "cover" the actual output resources. For reasons of conflict avoidance and delay minimization, it is desirable to have minimal

domains and ranges. Since transactions domains may be shared by different transactions, but typically transactions ranges may not, it is particulalry desirable to have minimal ranges. On the other hand, the price for increased granularity needed to achieve smaller ranges is an increased overhead in locking control.

3. Denote (x,z) an "oriented edge" where x is the origin and z is the target, and say that (x,z) is an "outcoming edge" with respect to node x, and an "incoming edge" with respect to node z. The "outdegree" or "indegree" of a node is the number of outcoming or incoming edges relative to the node. An outdegree zero node is called a "sink". An indegree zero node is called a "source". A "path" between a node x and a node z is a sequence of oriented edges (x,y1),(y1,y2),...,(yn,z) where n is greater or equal to one. A "cycle" is a path with z identical to x. A node z is "reachable" from a node x if there exists at least one path from x to z. A "subdigraph" s of a digraph g is a digraph obtained from g by suppressing zero or more nodes, each combination of incoming edge and outcoming edge (relative to the suppressed node) being replaced by an directed edge bypassing the suppressed node.

4. The set of sinks which are reachable from a transaction represented by node x is called the "prerequisite (transactions) set" of x and is denoted:

   PS(x)={z|z is a sink and there exists a path from x to z}

   A path from x to z where z is a sink is called a "prerequisite path".

For this section, let us assume that all the NVDBS modules have the power to lock/unlock their own resources, if any.

There is conceptually a "network concurrency control digraph" which could be maintained by some centralized (master) module of the NVDBS. Any cycle in such a digraph would correspond to a deadlock. Therefore, deadlock detection would boil down to the cycle detection in a digraph. Rather than maintaining such a digraph, we are proposing a distributed concurrency control scheme where each NVDBS component will maintain only its own relevant concurrency control digraph, referring only to these transactions which are interesting to the particular NVDBS component. Such concurrency control digraphs are subdigraphs of the global NVDBS concurrency control digraph, and somewhat similar to "projections" of the global digraph over the particular component.

A transaction x originates in an NVDBS module (probably an AGENT) denoted M[x]. From the viewpoint of M[x], the transaction x is ultimately blocked by PS(x), the prerequisite set of transactions for x in the control digraph of the module.

5.1.2 <u>Procedure</u> We assume that domains and ranges of a transaction are known ahead of transaction initiation time. In case of data dependent operations, this means unfortunately very large domains or ranges, which makes these operations undesirable.

A transaction t needs a resource r, at the module M[x]. If r is available, i.e. if no cycle exists in the M[x] control digraph, then all is well: if this is true for all r needed by t, the availability of r is updated in the resources tables, and the transaction t is initiated: it will not run into a deadlock since all needed resources have been reserved in advance.

If the requested r is not available, then since we are in a preventive mode, transaction t is prevented from proceeding until the resource becomes available. This means that the transaction will be wakened when all needed resources become available. Say in general that the resource r is held by a set of transactions (x1, x2, ..., xi). The control digraph is updated to include the oriented edges (t,x1), (t,x2), ..., (t,xi). The addition of some of these edges may give birth to a cycle in the control digraph, in which case the transaction t is rejected at this time. Rejection means no resources are set aside for the transaction, and the transaction is erased from all controls. Whoever originated the transaction may resubmit it again as if never rejected before.

If no cycle is detected, then transaction t is only temporarily blocked, as far as this particular module control digraph is concerned. The control of this module M[x] will then calculate PS(t). For each z in PS(t), the control of M[x] will then transmit to M[t] and M[z] the information that (t,z) is a prerequisite path. When (t,z) is received by a module, its control adds this oriented edge (t,z) to its digraph. If a cycle is thus formed, a deadlock is detected. If z is only blocked, then again, control will calculate PS(z), and for each z′ in PS(z), it will mail the prerequisite paths (z,z′) to M[z] and M[z′], if these are modules which are distinct from the current module.

Let us consider the following simple example: Initially, when t is proposed for execution at M[t], the following subdigraphs exist:

```
M[t] control subdigraph:      x1 ---> z
M[z] control subdigraph:      z  ---> z′
M[z′] control subdigraph:     z′ ---> t
```

Soon after t is proposed to M[t], the M[t] control digraph looks like this:

```
|---------|
|t ---> xl|---> z
|---------|
```

Where the box illustrates the temporary nature of the information. By now, M[t] calculates PS(t) and sends the first -- and only -- sink reachable from t: (t,z), to M[z]. M[z] control modifies its subgraph to the following:

```
|--------|
|t ---> z|---> z'
|--------|
```

Then M[z] finds z' as a reachable sink and mails (t,z') to M[z']. But M[z'] has now a cycle:

```
     --------->
z'             t
  |----------|
  |<---------|
  |----------|
```

M[z'] notifies M[t] of the rejection of t. All other involved modules must be notified of the fact that the temporary information is to be erased. That temporary information was evidenced by dashed boxes in the above figures.

By passing information linking directly a blocked transaction to the ultimate blocking transaction - rather than carrying the full prerequisite path - the propagation of indispensable information is performed at maximal speed, and deadlock discovery is swifter, while superfluous information is avoided in the control subdigraphs of the different NVDBS modules. The latter effect is to keep these subdigraphs to a minimum so that there is a minimum of information to keep up-to-date. The advantages of this very general approach are (a) no need for synchronized clocks, (b) only concerned modules are involved, (c) good scheme for high degree of granularity, to minimize the number of reachable nodes. The disadvantages are a huge communication overhead in case of numerous reachable nodes in the digraph, and sequential rather than parallel actions, thereby leading to greater delays.

When a hierarchy exists naturally among modules, and where geographic clusters exist, it may be beneficial to abandon the decentralized approach, and adopt a variant of the scheme as explained in [MENASCE78].

5.2 A DECENTRALIZED PROTOCOL BASED ON EVENTCOUNTS

In order to provide for a more efficient synchronization process, without demanding synchronized clocks, the concept of eventcount and ticket was developed, [GRAPA76] [LELANN78] [LAMPORT76] [REED76] [ELLIS77]. Each module maintains its own eventcounter in lieu of a synchronized clock. These eventcounters are carrying more or less the same count of elementary actions of the protocol, and are used very much like clocks to order sequences of actions.

5.2.1 Module; USER; SERVER Each module of the NVDBS is on a different host. A module is an AGENT, an NDM, or a REP. Each module has its own concurrency control process which involves a USER and a SERVER. The SERVER receives messages coming from USERs of other modules. The USER transmits messages to one or more modules (broadcast).

5.2.2 Eventcounter Each module concurrency controller maintains an "eventcounter". This counter contains the "eventcount", a number which is updated by every elementary protocol "events", as we shall see below. Such protocol "events" mark the new knowledge of another action or "phase" of the protocol for a given transaction. These events are the boundaries of the different "modes" or "phases" of the concurrency controller while executing the concurrency control protocol. In a sense, these eventcounters are clocks which count time in terms of different units: events rather than seconds. The eventcounter of a module plays the role of the synchronizing clock, vis-a-vis the different (network) transactions which involve common modules. One of the fundamental results of R&D studies in concurrency control is that a sufficient condition to avoid deadlocks between different transactions to be executed simultaneously is to prevent any younger competing transaction from being initiated.

The events are defined as follows:

- network transaction initial request.

- beginning of the proposition phase, which is evidenced by the receipt of a proposition message, which lets the receiving module know that a new transaction initial request was received and that the controller handling such initial request is now in the proposition phase for this transaction.

- beginning of the disposition phase, evidenced by the receipt of a disposition message, letting the receiving module that the controller for the particular transaction is now in the disposition phase.

As explained in [ELLIS77], each module's eventcounter is used to allocate a transaction number (t#) to each newly requested transaction initiated with the module. Each module has also a module number (m#) so as to break ties in a coherent manner, or to implement a virtual ring [LELANN78]. Every proposition message has a proposition number which is composed of the transaction number, the module number, and the attempted proposition number: (t#, m#, ap#). If the eventcount of the receiving module is less than the t# of the proposition received, then the eventcount is set equal to t#. Eventcounts may also be updated through any communication messages, as if they were carrying the "time" of the module they come from.

5.2.3 Basic mechanism of the protocol A given module controller is either in an idle mode, a proposition mode, or a disposition mode. A transaction is an NVDBS-transaction while a particular part of the transaction which pertains to a specific module would be a local query or local update. The protocol can be grossly summarized by the "state diagram" below:

```
|----------------------------------|
||---------|     |----------------||
||Idle mode|<--> |Proposition mode||
||---------|     |----------------||
|      ^             ^             |
|      |             |             |
|      V             V             |
|    |----------------|            |
|    |Disposition mode|            |
|    |----------------|            |
|----------------------------------|
```

Figure 63. State diagram

The idle mode is the mode of a module when no transaction is "NVDBS-processed" by the module. However, the module may still operate some local update pertinent to a transaction which originated in another module, or some other related action.

The Proposition mode is the mode resulting from a network transaction having been initiated in the module. In that mode, the module will analyze the transaction and propose to all other relevant modules the specific local updates or local queries entailed by the transaction. But in that mode too, the module may allow certain interruptions.

The disposition mode, finally, is the mode of the module when the transaction processed by the module is ready for execution. This follows from the acceptance (of all involved modules) of the proposed local actions entailed by the transaction. This mode allows certain interruptions.

As a rule, the different possible "transitions" or "events" which can occur in a given mode may not themselves be interrupted. The following paragraphs give some more details about the three different modes.

5.2.3.1 <u>Idle mode</u> In the idle mode, a controller can be interrupted by three sorts of messages: a proposition, a disposition, or an initial network transaction request. In the first two cases, the controller acts only as a SERVER, e.g. a REP's controller would act in that fashion. In the last case, the typical example is the AGENT's controller, receiving a network transaction request.

```
|-----------------------------------------------------------------|
|Idle mode         Proposition                                    |
|                  received from                                  |
|     |            proposing module                               |
|     |                     |                                      |
|     |                     |                                      |
|     V                     V                                      |
|---------------------------------> transmit acceptance acknowledgement|
| update eventcounter & lock        or rejection acknowledgement back |
|               |                   to proposing module           |
|               |                                                 |
|               V                                                 |
|          Idle mode                                              |
|-----------------------------------------------------------------|
```

Figure 64. Idle mode/proposition interrupt

```
|-------------------------------------------------------------|
|Idle mode       disposition message                          |
|    |           i.e. specific signal                         |
|    |           to execute an update                         |
|    |           previously proposed                          |
|    |           and accepted                                 |
|    |                       |                                |
|    |                       |                                |
|    V                       V                                |
|------------------------------------> transmit DONE to the disposing|
|perform the actual local update                       module.|
|               |                                             |
|               |                                             |
|               |                                             |
|               V                                             |
|        Idle mode                                            |
|-------------------------------------------------------------|
```

Figure 65. Idle mode/disposition interrupt

If M is the number of modules which acknowledge the proposition, the transition from idle mode to proposition mode looks like this:

```
|-----------------------------------------------------|
|Idle mode      original network transaction          |
|    |          request, or transaction initiation    |
|    |                      |                         |
|    V                      V                         |
| ------------------------------->|broadcast proposition |
| update eventcounter & lock      |to all N involved   |
| allocate t# and set M=0         |modules             |
|               |                                     |
|               |                                     |
|               V                                     |
|       Proposition mode                              |
|-----------------------------------------------------|
```

Figure 66. Idle mode/event = transition to proposition mode

5.2.3.2 Proposition mode Five basic events may occur in that mode:

```
|--------------------------------------------|
|Proposition mode    Acceptance message      |
|       |            from "proposed" module  |
|       |                      |             |
|       |                      |             |
|       V                      V             |
|----------------------------------------    |
|               set M = M + 1                |
|                     |                      |
|                     |                      |
|                     V                      |
|            Proposition mode                |
|--------------------------------------------|
```

Figure 67. Single acceptance in proposition mode

```
|--------------------------------------------------------------|
|Proposition mode    Rejection                                 |
|       |            relative to                               |
|       |            current transaction                       |
|       |                      |                               |
|       V                      V                               |
|----------------------------> transmit rejection to requestor |
|            |                 and broadcast cancellation to   |
|            |                 modules already involved        |
|            V                                                 |
|        Idle mode                                             |
|--------------------------------------------------------------|
```

Figure 68. proposition mode/receipt of rejection of current transaction

```
|------------------------------------|
|Proposition mode    Rejection       |
|       |            relative to     |
|       |            old transaction |
|       |            already rejected|
|       |                  |         |
|       V                  V         |
|------------------------------------|
|(the rejecting module was not told  |
| in time of the cancellation)       |
|                  |                 |
|                  |                 |
|                  V                 |
|       Proposition mode             |
|------------------------------------|
```

Figure 69. proposition mode/receipt of rejection of old transaction

```
|-------------------------------------------------------|
|Proposition mode        M=N-1                          |
|       |                all "propositioned"            |
|       |                modules have now               |
|       |                answered favorably             |
|       |                          |                    |
|       V                          V                    |
|-----------------------------------> accept original   |
|                |                    transaction request|
|                |                                      |
|                V                                      |
|        Disposition mode                               |
|-------------------------------------------------------|
```

Figure 70. proposition mode/ transition to disposition mode

```
|------------------------------------------------------------|
| Proposition mode       Proposition                         |
|        |               relative to                         |
|        |               different                           |
|        |               transaction                         |
|        |                    |                              |
|        V                    V                              |
| ------------------------------------                       |
| update eventcounter and                                    |
| test to know if the transactions                           |
| require some common resources                              |
|    |                             |                         |
|    |no                           | yes                     |
|    V                             |                         |
|---------> accepting              |                         |
|    |      acknowledgement        |                         |
|    V                             |                         |
|Proposition mode                  |                         |
|                                  V                         |
|               ------------------------------------          |
|               is eventcount of original transaction        |
|               less than eventcount of proposed one?        |
|                |no          |yes                           |
|                |            V                              |
|   interrupt|           ----------> negative acknowledgement |
|   by higher|                |       sent to "proposing" module |
|   priority |                V                              |
|   task     |           proposition mode                    |
|                V                                           |
|      ---------------->|rejection sent to original transaction|
|                |      |requestor; cancellation to all modules|
|                |      |involved in old transaction; acceptance
|                V      |sent to propositioning module.       |
| Idle mode (under proposition, see Fig. 51)                 |
|------------------------------------------------------------|
```

Figure 71. proposition mode/interrupting proposition

5.2.3.3 <u>Disposition mode</u> There are three basic events in the disposition mode:

```
|------------------------------------------------------------------------|
| disposition mode/                                                      |
| disposition state                                                      |
|        |                                                               |
|        |                                                               |
|        V                                                               |
| ------------------>  broadcast disposition orders to relevant modules|
|                                                                        |
| (1) perform local update (if any)                                      |
|     relevant for original transaction                                  |
| (2) set M=0                                                            |
|        |                                                               |
|        |                                                               |
|        V                                                               |
| disposition mode/                                                      |
| completing state                                                       |
|------------------------------------------------------------------------|
```

Figure 72. disposition mode/disposition broadcast

```
|----------------------------------------|
|disposition mode/                       |
|completing state          "Done" message|
|      |                          |      |
|      |                          |      |
|      V                          V      |
|   ---------------------------------    |
|                  |                     |
|                  | M=M+1               |
|                  V                     |
|disposition mode/ completing state      |
|----------------------------------------|
```

Figure 73. disposition mode/ single completion

```
|---------------------------------------------------|
|disposition mode/        interrupt by              |
|completing state         Disposition message       |
|      |                  relative to other         |
|      |                  transaction               |
|      |                         |                  |
|      V                         V                  |
|--------dispose of update--------> "Done" sent back|
|                   |                               |
|                   |                               |
|                   V                               |
|disposition mode/ completing state                 |
|---------------------------------------------------|
```

Figure 74. disposition mode/ disposition interrupt

```
|------------------------------------------------------------------|
|disposition mode/          M=N number of concerned modules        |
|completing state           all "Done" have been                   |
|      |                    received, including this module        |
|      |                    itself|                                |
|      |                          |                                |
|      V                          V                                |
|----------------------------------> send "Completed" to original|
|                  |                   NVDBS transaction requestor |
|                  |                                               |
|                  V                                               |
|             Idle mode                                            |
|------------------------------------------------------------------|
```

Figure 75. disposition mode/ final completion

5.3 <u>DETAILS ON AN NVDBS DECENTRALIZED CONCURRENCY CONTROL PROTOCOL</u>

We now present a solution which assumes the availability of timestamps to the different modules of the NVDBS. This assumes a fully synchronized underlying computer network, a feasible feature for a local network. The terminology of the previous section is still valid. Let us discuss some details of the proposition and disposition phases.

5.3.1 <u>Proposition phase</u> Proposition initiation:

```
|--------------------------------------|
|                 |------> (a-1) AGENTs|
|                 |                    |
|proposing AGENT -|                    |
|                 |                    |
|                 |------> r REPs      |
|--------------------------------------|
```

Figure 76. proposition initiation

Following the above figure, the proposing AGENT sends a "proposition" to all other AGENTs and to all REPs in the NVDBS.

A. All REPs which are involved in the proposed transaction are called the "involved REPs". These involved REPs are given their particular subtransaction, and they are asked to indicate which type of control action they are taking and/or they would take. Two kinds of accesses are typically contemplated:

   a. READ (only) which can be shared with other READ accesses, but locks out any update. This is the normal access for the transaction "domain" -- i.e. input or required resources.

   b. UPDATE (exclusive) which locks out any other access: this is the type of access for the range of the transaction (output resources).

We say that a transaction at the AGENT level is decomposed into a number of READs and UPDATEs on different REPs. A particular (involved) REP will be proposed a READ or an UPDATE.

a. If it is a READ, the REP must check that (1) all "preceding" UPDATE propositions or dispositions (from "preceding" transactions, with older timestamps) have been received, and that (2) among these, all "relevant" UPDATE dispositions -- and here "relevant" means pertinent to the same database element (*) -- have all been executed, in the timestamps order of their transactions.

b. If it is an update, it can be executed only after all the prerequisite READs of the same transaction have been executed by their REPs and these REPs have sent their UPDATE disposition messages to the REP which is to execute this dependent UPDATE.

B. Concurrency control and resiliency: all AGENTs and all REPs are requested copies of any unacknowledged messages to the involved REPs, as well as copies of all rejected transaction(s)/ proposition(s). They are all given the timestamp of the (proposed) transaction. They are also given the list of transactions rejected by the proposing AGENT. This is for the purpose of updating the concurrency controllers of the AGENTs and REPs.

* a database element or entity is the smallest portion of database on which a lock/access mechanism is applied. Note that this may be DBMS dependent.

C. Transaction synchronization: all "uninvolved" REPs are warned to send outstanding -- if any -- UPDATE messages (disposition messages) of other transactions to the relevant involved REPs of the proposed transaction.

Proposition acknowledgement: The acknowledging NVDBS module sends back the following information:

1. proposition acknowledgement

2. list of all UPDATE messages - not yet sent - intended for involved REP, and left over from previous transaction;

3. list of all UPDATE messages which are going to be generated, and otherwise like above;

4. list of all transactions which were rejected by the acknowledging NVDBS module;

5. If the acknowledging module is an involved module, it acknowledges also its particular instructions relative to the proposed transaction;

5.3.2 Disposition phase All NVDBS modules have answered the proposition satisfactorily. The proposing module (typically an AGENT) now broadcasts disposition commands to the READ involved REPs, in order to start execution. All such REPs must first execute all previous transactions remaining UPDATEs. This constraint linearizes the sequence of actions and prevents deadlock. The READ involved REPs execute their READ subtransactions and then send their UPDATE messages to the UPDATE involved REPs for the transaction final disposition.

5.3.3 Robustness considerations One NVDBS module sends a message to another module:

```
|------------------------------------------------------|
| |------------------|        |---------------------| |
| |sending site's USER|------->|receiving site's SERVER| |
| |------------------|        |---------------------| |
| (keeps a copy of any message                         |
|  it sends until acknowledged)                        |
|------------------------------------------------------|
```

Figure 77. USER->SERVER

non responding modules are "timed out". The remaining modules are "in". If there is at least one copy of the needed domain data, or data to be read, the transaction proceeds, in spite of the possibility that some range REPs may have been "timed out". Logging of all messages becomes mandatory when some involved SERVER is timed out, and the USER-SERVER messages propagate the warning that the transaction is contaminated by the possibility of an ulterior recovery. Other subsequent transactions may be contaminated too.

Before a USER initiates "disposition phase" by sending its READ messages, it has determined that all the necessary READs have responsive SERVERs, and if some UPDATE modules SERVERs are not responsive, it has already included precaution warnings in the READ messages in order to have every modules backup its messages and related data.

This propagation functions as a "contamination". When a SERVER is timed in, a "reconciliation process" takes place. A number of transactions may have to be "undone" and "redone". In general, unexecuted subtransactions (READs or UPDATEs) are untolerable. The exception is when READs or UPDATEs could be executed on at least one of several redundant copies of the data, in which case redundancy is used to make the NVDBS more resilient to potential down time of SERVERs.

First, a transaction involving only READs, would not raise any problem. Also, if the transaction can reach at least one redundant copy of every data item on which it must execute a READ or UPDATE, there is always a chance that the other copies may be correctly updated later on.

A transaction is either accepted or rejected. It is accepted if all READ involved SERVERs are in, rejected otherwise - unless all non redundant data is in.

If rejected, the transaction will propagate a "precaution warning" in the case when some involved SERVERs have been timed out.

5.4 CONCLUSIONS ON CONCURRENCY CONTROL

Clearly, the ideal candidate DBMS to be branched on the NVDBS is one whose concurrency control mechanism is a preventive scheme. But the problem of recovery caused by other malfunctions remain. The REP transactions could be backed up by the DBMS. The problem is that an NVDBS transaction might have read the wrong thing and will never know it, as a price for the NVDBS principle #1.

In general, the power of the ordinary user to lock or tie up certain resources of the DBMS would be indispensable for the NVDBS to implement an efficient concurrency and recovery control.

Although the academic aspects of the concurrency control problem are fascinating, it does not follow that the importance of the problem with respect to the overall NVDBS design is of the same magnitude. Indeed, updates should not be too frequent on the network, and the granularity of the database should be refined: in view of the communication expense, the locking control overhead is not an overriding concern, and one should minimize the risk of tieing up large resources when communications are bad, thereby lowering the NVDBS interference on the local DBMS.

In conclusion, the decentralized scheme should be implemented until geographic and update considerations suggest a compromise [ALSBERG] between centralized and decentralized concurrency control. Although there is some useful knowledge to be gained by the study of the synchronization problem in operating systems, the DBMS synchronization problem is vastly different, as pointed out in [SCHLAGETER78]. Indeed, DBMS processes work on a large and variable number of resources, and these resources are interdependent (e.g. consistency constraints); these resources are created, updated and deleted by the processes, and they may be addressed by contents, or associatively rather than just be named. In this context, it becomes apparent that simulation studies and other practical experience are invaluable.

6. REFERENCES

- [ALSBERG76] "A principle for resilient sharing of distributed resources" Center for advanced computation, U. of Illinois, Urbana, 1976.

- [ALSBERG78] panel discussion, third Berkeley workshop on distributed data management and computer networks, 1978.

- [BACHMAN69] "Data Structure Diagrams" DATA BASE, Vol.1, No.2, ACM SIGBDP, 1969.

- [BADAL78] "A proposal for distributed concurrency control for partially redundant data base systems" D. Z. Badal, G. J. Popek, Third Berkeley Workshop on distributed systems and computer networks, 1978.

- [BLASGEN76] "On the Evaluation of Queries in a Relational System" y Michael W. Blasgen and Kapali P. Eswaran, IBM Research Laboratory, San Jose, Ca. RJ1745(#25553), 8 April 1976.

- [BOCHMANN78] "Synchronization in distributed system modules" Gregor V. Bochmann, Third Berkeley Workshop on distributed systems and computer networks, 1978.

- [CATIS76] "Computer-Aided Tactical Information System Section IX - Data Base Structure" by Bunker Ramo, pages 9-1 thru 9-28.

- [CATIS77] "Computer-Aided Tactical Information System (CATIS) Software System Design Document," by Bunker Ramo, Revision C, change 1, April 1977.

- [CHU76] "Performance of File Directory Systems for Data Bases in Star and Distributed Networks", by W. W. Chu, NCC76, AFIPS Proceedings Vol. 45, pp577-587, 1976.

- [CODASYL-DBTG71] CODASYL Programming Language Committee, Data Base Task Group Report, available from ACM, April 1971.

- [CODD74] "Seven Steps to Rendezvous with the Casual User", Proc. IFIP TC-2 Working Conference on Data Base Management Systems, Cargese, Corsica, April 1-5, 1974, North-Holland.

- [CODD75] "Recent Investigations in Relational Data Base Systems" E. F. Codd, IBM Research Laboratory, Monterey and Cottle Roads, San Jose, California, 95193; ACM PACIFIC 75, April 17-18, San Francisco.

- [CODD/DATE74] "Interactive Support fro Non Programmers: The Relational and Network Approaches" by E. F. Codd and C. J. Date, June 6 1974, RJ1400, IBM Research Center, San Jose, Ca. 95193.

- [DATE75] "An introduction to Database Systems" by C. Date, Addison Wesley.

- [DATE/CODD74] "The Relational and Network Approaches: Comparison of the Application Programming Interfaces" by C. J. Date and E. F. Codd, June 6 1974, RJ1401, IBM Research Center, San Jose, Ca. 95193

- [ELLIS77] "A robust algorithm for updating duplicate databases" by Clarence Ellis, Second Berkeley Workshop on distributed data management and computer networks, May 25-27 1977.

- [ELLIS77] "Consistency and correctness of duplicate database systems" by Clarence A. Ellis, May 1977, Xerox Palo Alto Research Center Report.

♦ [ESWARAN76] "The notions of consistency and predicate locks in a database system" by K. P. Eswaran, J. N. Gray, R. A. Lorie, I. L. Traiger, CACM Nov. 76, pp624-633.

♦ [FERNANDEZ75] "An authorization model for a shared data base" by E. B. Fernandez, R. C. Summers, and C. D. Coleman, IBM Los Angeles Scientific Center. ACM-SIGMOD 75, pp23-31.

♦ [GARCIA78] "Distributed Database Coupling" by Hector Garcia Molina, Third USA-Japan Computer Conference, Oct. 10-12 1978, San Francisco.

♦ [GOTLIEB75] "Computing Joins of Relations" ACM-SIGMOD 1975, San Jose, Ca., May 1975, pp55-63.

♦ [GRAPA76] "Characterization of a distributed database system" Technical report UIUCDCS-R-76-831, Dept. of CS, Univ. of Illinois, Urbana, Oct. 76.

♦ [GRAY78] panel discussion, third Berkeley workshop on distributed data management and computer networks, 1978.

♦ [IMS] IBM Corporation's Information Management System / Virtual Storage, Utilities Reference Manual, IBM no. SH20-9029.

♦ [INCO76] "Terminal Transparent Display Language", INCO, Inc, Va., Jan 76 AD/A-021 331

♦ [INGRES76] "The Design and Implementation of Ingres" Michael Stonebraker, Eugene Wong, Peter Kreps, Gerald Held; Mem.No. ERL-M577, 27 Jan. 1976. College of Engineering, University of California, Berkeley, California, 94720.

♠ [INGRES77] "Ingres Reference Manual" Mem.No. ERL-M579, 20 Dec. 1977. College of Engineering, University of California, Berkeley, California, 94720.

♠ [INGRES78] "Distributed Query Processing in a Relational Data Base System" R. Epstein, M. Stonebraker, E. Wong. SIGMOD 78, ACM.

♠ [LAMPORT76] "Time, Clocks and the ordering of Events in a distributed system", Massachussetts Computer Associates Report, CA-7603-2911, March 1976.

♠ [LEHOT76] "On the Optimal Ordering for an Indexed Sequential File Organization" by Lehot et al., Proceedings of the Tenth Asilomar Conference on Computer Systems and Circuits, Nov. 1976.

♠ [LEHOT78] "User Directed Data Base Design" by Philippe Lehot, Eleventh Hawaii International Conference on System Sciences, Jan. 78, Honolulu, HI.

♠ [LELANN78] "Algorithms for distributed data sharing systems which use tickets", Gerard LeLann, Third Berkeley Workshop on distributed database systems and computer networks, 1978.

♠ [LORIE77] "The compilation of a very high level data language", by Raymond A. Lorie and Bradford W. Wade. IBM Research Laboratory, RJ2008 (28098) 5/27/77. San Jose, Ca. 95193.

♠ [LORIE78] "An Access Specification Language for a Relational Data Base System" by Raymond A. Lorie and Jorgen F. Nilsson. IBM Research Laboratory, San Jose, Ca. 95193. RJ2218(30171)4/11/78 Computer Science.

- [LUKES74] "Efficient Algorithm for the Partitioning of Trees" IBM Journal of Research and Development, Vol 18, May 1974.
- [MARTIN75] "Computer Data-Base Organization" by James Martin, Prentice Hall.
- [McCAULEY77] "An Introduction to the Network Virtual Data Base" Ford Aerospace and Communications Corporation, WDL Division, 3939 Fabian Way, Palo Alto, California, 94303.
- [MENASCE78] "Locking and deadlock detection in distributed databases" by Daniel A. Menasce and Richard R. Muntz, Third Berkeley Workshop on distributed database systems and computer networks, 1978.
- [MINOURA78] "Maximally Concurrent Transaction Processing" by Toshimi Minoura, Third Berkeley Workshop on distributed database systems and computer networks, 1978.
- [NIJSSEN76] "A gross architecture for the next generation data base management systems" by G. M. Nijssen, in "Modelling in Data Base Management Systems, IFIP TC2, Freudenstadt, 1976, North Holland, Amsterdam, Nijssen ed.
- [NUTT72] "Evaluation Nets for Computer Performance Analysis" Proceedings of FJCC 1972.
- [OASIS76] "USAFE Operational Applications of Special Intelligence System in the Central Region (OASIS/CR), March 1976.
- [PEEBLES78] "System Architecture for Distributed Data Management" by Richard Peebles and Eric Manning, Computer, Vol. 11, no. 1, January 1978,

page 43.

♦ [PADLIPSKY76] "A Perspective on Intercomputer Networking" by Micheal A. Padlipsky, an internal MITRE-Bedford document, 1976.

♦ [PETERSON77] "Petri Nets" ACM Computing Surveys, Vol. 9, No. 3, September 1977.

♦ [QIP77] "Transparent Integrated Intelligence Network Query Intermediate Processor," by INCO, Incorporated for RADC, Final Technical Report RADC-TR-77-39, January 1977, A037947.

♦ [REED76] "Synchronization with eventcounts and sequencers" by D. P. Reed and R. K. Kanodia, MIT Report 1976.

♦ [RUBY78] "An Approach to Providing a Relational Query Processor for a limited CODASYL Data Base Management System" by James B. Ruby and A. G. Carrick. Third USA-Japan Computer Conference, Oct. 10-12 1978, San Francisco.

♦ [SARP77a] "SARP IV User's Manual" by Bunker Ramo, a rough draft written September 1977.

♦ [SARP77b] "SARP IV Development Guide RTSS Advanced Operating Capabilities Storage and Retrieval Processor (SARP IV)" by Bunker Ramo.

♦ [SARP77c] "SARP File Description," Appendix A thru J of SARP IV User's Manual, by Bunker Ramo.

♦ [SCHKOLNICK77] "A Clustering Algorithm for Hierarchical Structures" ACM Transactions on Database Systems, March 1977, Vol 2, no 1, pages 27-44.

♦ [SCHLAGETER78] "Process synchronization in Database systems" ACM Transactions on Database Systems, September 1978, Volume 3, Number 3.

♦ [SEQUEL76] "SEQUEL2: A Unified Approach to Data Definition, Manipulation and Control" by D. D. Chamberlin, M. M. Astrahan, K. P. Estrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner and B. W. Wade. IBM Journal of Research & Development. Vol. 20, No. 6, Nov. 76.

♦ [SMITH75] "Optimizing the Performance of a Relational Algebra Database Interface" by J. Smith and P. Chang, CACM, October 1975.

♦ [SSB78] "Standard Software Base (SSB), Release III" by INCO, Incorporated, Final Technical Report # INCO/1073-378-tr-26-D(F), 28 February 1978.

♦ [SUMMERS75] "A Programming Language Extension for Access to a Shared Data Base" by Summers, Coleman and Fernandez. ACM PACIFIC 75.

♦ [SYSTEM R] "System R: Relational Approach to Database Management" Astrahan et al., ACM TODS, Vol. 1, No. 2, June 1976, Pages 97-137.

♦ [STONEBRAKER78] "Concurrency control and consistency of multiple copies of data in distributed ingres" by Michael Stonebraker, Third Berkeley Workshop on Distributed Database Systems and Computer Networks, 1978.

♦ [S2000/74] "System 2000: Reference Manual" Copyright 1974, MRI Systems Corporation, 12675 Research Boulevard, Austin, Texas, 78766.

♦ [TCP78] "Internetwork Protocol Specification, Version 4" by Jonathan B. Postel, Internetwork Experiments Notebook 54, section 2.3.2.1, September 1978.

- [TCP78] "Specification of Internetwork Transmission Control Protocol, Version 4" by Jonathan B. Postel, Internetwork Experiments Notebook 55, section 2.4.2.1, September 1978.

- [THOMAS75] "A psychological study of query by example" by John C. Thomas and John D. Gould, NCC75, pp439-446.

- [TSICHRITZIS77] "Data Base Management Systems" by Dionysios C. Tsichritzis and Frederick H. Lochovsky, Academic Press, 1977.

- [TTDL76] "Terminal Transparent Display Language" by INCO, Incorporated, U.S. Department of Commerce National Technical Information Service document AD/A-021 331, January 1976.

- [WIANT74] "A Partitioning Algorithm for Hierarchic Data Structures to Minimize Resource Consumptions" by J. E. Wiant, IBM Palo Alto J24/034. Palo Alto, Ca.

- [WIEDERHOLD77] "Database Design" by Gio Wiederhold, McGraw-Hill.

- [WONG76] "Decomposition - A Strategy for Query Processing" by Eugene Wong and K. Youssefi. ACM Transactions on Database Systems, vol. 1, no. 3, September 1976, pp.223-241.

- [WONG77] "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases", by Eugene Wong. Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 25-27, 1977.

- [YEH75] "Modeling of concurrent control structures and parallel processing" Ph.D. Thesis, University of Maryland, 1975.

♦ [YAO78] "Query Processing on a Distributed Database" Third Berkeley Workshop on Distributed Data Management and Computer Networks, 1978.

♦ [ZLOOF75] "Query by example", Moshe Zloof, NCC75, pp431-438.

♦ [ZLOOF76] "Query by example: operations on the transitive closure" by Moshe Zloof, Tenth Asilomar Conference on Circuits, Systems, and Computers, Nov. 1976. pp 256-261.

♦ [ZLOOF77] "The System for Business Automation (SBA): Programming Language" CACM June 1977, Vol. 20, No. 6, pp385-396.

♦ [ZLOOF77] "Query-By-Example: a data base language", by Moshe Zloof, IBM System Journal, No. 4, 1977, pp 324-343.